On the Fly Detection of the Top-k Items in the Distributed Sliding Window Model

Emmanuelle Anceaume, Yann Busnel, Vasile Cazacu

On the Fly Detection of the Top-k Items in the Distributed Sliding Window Model

Emmanuelle Anceaume
CNRS, IRISA
Rennes, France
Email: emmanuelle.anceaume@irisa.fr

Yann Busnel
IMT Atlantique, IRISA
Cesson-Sévigné, France
Email: yann.busnel@imt-atlantique.fr

Vasile Cazacu
CNRS, IRISA
Rennes, France
Email: vasile.cazacu@irisa.fr

***Abstract*—This paper presents a new algorithm that detects on the fly the k most frequent items in the sliding window model. This algorithm is distributed among the nodes of the system. It is inspired by a recent and innovative approach, which consists in associating a stochastic value correlated with the item's frequency instead of trying to estimate its number of occurrences. This stochastic value corresponds to the number of consecutive heads in coin flipping until the first tail occurs. The original approach was to retain just the maximum of consecutive heads obtained by an item, since an item that often occurs will have a higher probability of having a high value. While effective for very skewed data distributions, the correlation is not tight enough to robustly distinguish items with comparable frequencies. To address this important issue, we propose to combine the stochastic approach together with a deterministic counting of items. Specifically, in place of keeping the maximum number of consecutive heads obtained by an item, we count the number of times the coin flipping process of an item has exceeded a given threshold. This threshold is defined by combining theoretical results in leader election and coupon collector problems. Results on simulated data show how impressive is the detection of the top-k items in a large range of distributions.**

I. Introduction

The need to analyze in real time large-scale and distributed data streams has recently became tremendous to detect attacks, anomalies or performance issues. In particular the identification of recent heavy-hitters (or *hot* items) is essential but highly challenging. This problem has been heavily studied during the last decades with both exact and probabilistic solutions [1], [2]. A great survey and empirical comparisons of available and most used techniques were done by Cormode and Hadjieleftheriou [3]. While simple to state and fundamental for advanced analysis, answering this issue over a time sliding window [4], [5] and among distributed nodes [6] is still an active research field. The distributed detection of frequent items over a sliding window presents two extra challenging aspects with respect to the centralized detection of frequent items since the inception of the stream:

- Treat time decaying items as they enter and exit the sliding window;
- Produce mergeable local stream summaries in order to obtain a system-wide summary.

Very recently, Song, Liu and Ge [7] formalized this problem as the windowed top-k frequent items (WTK) problem and proposed an efficient and very elegant solution, named the floating top-k (FTK) method, to solve WTK. In a nutshell, instead of counting items or estimate their frequency, FTK associates a stochastic value correlated with the item's frequency in order to identify the most frequent ones.

Our contributions. We improve upon their solution by providing a new algorithm called FTK$_{CE}$. It is based on a deterministic counting of the most over-represented items in the data streams, which are themselves probabilistically identified using a dynamically defined threshold. Performance of our new algorithm are astonishingly good, despite any items order manipulation or distributed execution[1].

Paper roadmap. The remaining of our paper is organized as follows. Works related to our problem are presented in Section II. Section III provides a specification of the problem addressed in this paper. Section IV is dedicated to our algorithm description, and Section V details the construction of the algorithm's threshold. Performance evaluation is presented in Section VI. Finally, Section VII concludes this paper and presents some future works.

II. Related Works

A lot of research works have been dedicated to the detection of top-k items in continuous and massive streams since the seminal work of Misra and Gries [8] in 1982. As detailed shortly, lines of work particularly focus on the space and time complexity of the detection algorithms in order to cope with the continuously increasing rates of data generation. Some works also address orthogonal features related to the semantic of the streams by providing the possibility to answer queries on the recent past of a particular stream or on distributed ones [3]. Research in the detection of top-k items can be classified in two groups: counter- and sketch-based algorithms.

A. Sketch-based techniques

The two most known, versatile and general purpose sketch summaries are CountSketch (also known as AMS sketch) due to Alon, Matias and Szegedy [9] and CountMin sketch due to Cormode and Muthukrishnan [2]. Actually, both techniques are used to extract the number of times (also called the frequency) each item occurs in a stream. Such an estimation is done by relying on the properties of the linear projection

[1]Both algorithms, FTK$_{CE}$ and FTK, share these two properties (cf. Section IV-D).

(using universal hash functions) of the initial data space onto a smaller space of counters. In order to mitigate inevitable collision of items, several pairwise-independent hash functions are used in parallel. The top-k items are then extracted from the sketch data structure.

B. Counter-based techniques

One of the most famous counter based technique is undeniably the SPACESAVING algorithm by Metwally, Agrawal et El Abbadi [10]. The underlying idea of SPACESAVING is just to store a limited number of counters. The associated counter is incremented when an already seen item arrives, and all counters are decremented when a new item arrives and all counters are already taken. While initially thought to answer the frequent and the top-k items problems, SPACESAVING can be extended to answer the frequency estimation problem too.

C. Mixed techniques

Recent works have mostly been focused on improving, extending or mixing precedent works as achieved for example in [5], [4], [6], [11], [12], [13]. A common idea of both [5] and [13] to enhance the performance of SPACESAVING consists in filtering out some of the items (i.e., the ones supposed to have low frequencies) before applying SPACESAVING by adding a hashed bitmap counter for [5] and two COUNTMIN sketches for [13].

D. Probabilistic counting technique

Almost all of the previous works are based on improvements and/or combinations of counting methods and hashing techniques. Differently, in 2017, Song et al. [7] presented a new type of algorithm to directly identify the top-k most frequent items without counting or estimating their frequencies. The core idea of their proposal is to generate a stochastic value every time an item arrives and just to keep the maximum value obtained by different items. The logic behind this technique is that more often an item occurs, more numerous are its chances to get a higher value. The estimated top-k most frequent items are given by the items with the k highest values. Finally, this innovative approach presents two very interesting properties:

- The algorithm is resilient to the permutation of items within the data stream, and
- A distributed sliding window schema can be derived straightforwardly.

These points will be discussed in Section IV.

III. PROBLEM STATEMENT

In this section we discuss some important features of the sliding window and distributed version of the top-k most frequent items problem.

A. Terminology

Sliding window. There are mainly two ways to define a sliding window when treating a data stream. A sliding window is either defined by the last n_W received items or by the items received during the last W units of time. We adopt the second definition, since it matches a larger range of applications. Specifically, the time domain is partitioned into *time units* (e.g., seconds, hours, days, etc.), which define the time granularity of the sliding window over the data stream. Indeed, the time unit acts like a *micro batch* for the windowed top-k most frequent items problem: the top-k most frequent items are extracted at each time unit, after what the top-k most frequent items over the whole past time window is returned. The choice of the time unit is completely arbitrary and has to be defined by the user and depends on the problem at hand.

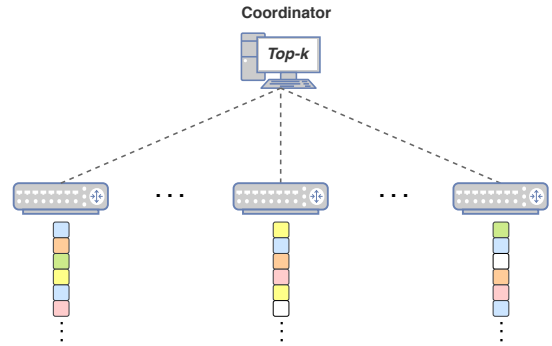


Fig. 1. System model

Thresholded counts, heavy hitters, top-k. The thresholded counts problem (e.g., [14]) aims to identify all the items that crossed a previously defined threshold. Since this threshold is independent of the stream length, this can lead to a potentially unbounded number of different items crossing this threshold. On the other hand, heavy-hitters are items whose frequency exceeds a given proportion ϕ of the stream size (e.g., [15]). There can be at most $1/\phi$ heavy hitters at any time or during one sliding window. The motivation behind the top-k most frequent items problem is to avoid when possible the use of a statically defined threshold, which in highly dynamic and large data streams can produce a very fluctuating number of heavy hitters since ϕ is likely to be relatively low. In addition, there are many use cases (e.g., social networks hot topics, marketing applications, etc.), where we are only interested in the top-k most frequent items whatever the relative frequency of each item. Section III-B formally defines the top-k most frequent items.

B. System Model

We consider a set of $\mathcal{N}$ nodes $S_1, \ldots, S_{\mathcal{N}}$ such that each node S_i receives a large sequence σ_{S_i} of data items. In the following we use the generic term "item" to represent any received element or symbol. We assume that streams $\sigma_{S_1}, \ldots, \sigma_{S_{\mathcal{N}}}$ do not necessarily have the same size, i.e., some of the items present in one stream do not necessarily appear in others or their number of occurrences may differ from one stream to another one. We also suppose that node S_i ($1 \leq i \leq \mathcal{N}$) does not know the length of its input stream. Items arrive at high rates and due to memory/resource constraints, they need to be processed sequentially and in an online manner. Indeed, nodes can locally store only a small

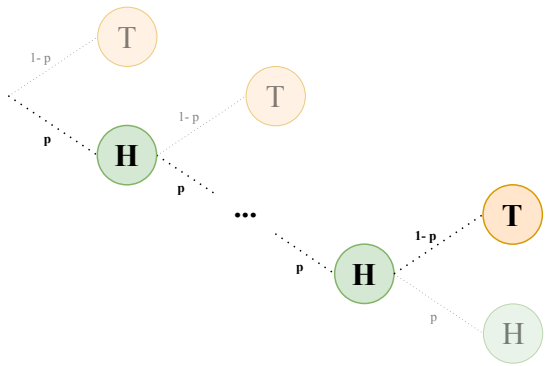


Fig. 2. Coin tossing until first tail.

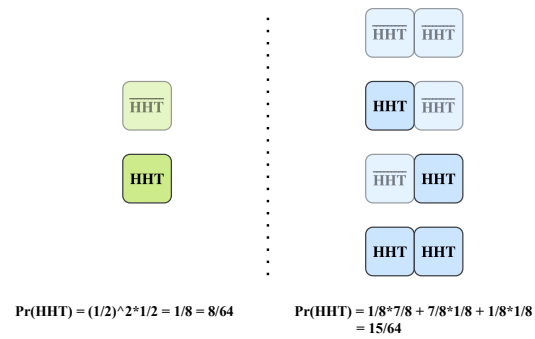


Fig. 3. Obtaining exactly two consecutive heads with one or two trials.

amount (i.e., a fraction) of the information with respect to the size of their input data streams and only perform simple operations on them (e.g., elementary arithmetic operations).

Moreover, we suppose that nodes $S_1, \ldots, S_{\mathcal{N}}$ cannot communicate among each other. Thus we suppose the existence of a specific node, called the *coordinator* in the following, with which each node is allowed to communicate (see Figure 1). A coordinator is there to avoid naive centralization of all data and to significantly reduce communication cost over the network. We assume that the communication is instantaneous.

Let $\sigma = a_1, a_2, a_3, \ldots, a_n$ be a data stream of items that arrive regularly and sequentially. Each data item i is drawn from the universe $\Omega = \{1, 2, \ldots, N\}$, where N is very large. A natural approach to study a data stream σ of length n is to model it as a fingerprint vector over the universe Ω, given by $F = (f_1, f_2, \ldots, f_N)$ where f_i represents the number of occurrences of item i in σ and $0 \leq f_i \leq n$. Note that in the following by abusing the notation, we denote this "$|\Omega|$-point distribution" by "Ω-point distribution", also known as the item frequency vector of σ.

C. Problem statement

We now formalize the top-k frequent items problem, the windowed top-k frequent items one, and the distributed version of the windowed top-k frequent items problem.

Definition 1 (Top-k Frequent Items)**.** *Given a data stream σ of n items $a_1, a_2, a_3, \ldots, a_n$, the top-$k$ most frequent items over σ is the set $\{i \in \Omega \mid f_i \geq f_{(k)}\}$ where $f_{(k)}$ is the k-th greatest value in F.*

Definition 2 (Windowed Top-k Frequent Items (WTK) [7])**.** *For any time t and any number of time units W, the windowed top-k frequent items problem consists in returning the top-k most frequent items received from time $t - W$ to time t.*

Definition 3 (Distributed Windowed Top-k Frequent Items)**.** *For any time t, any number of time units W and $\mathcal{N}$ nodes, $S_1, \ldots, S_{\mathcal{N}}$, the distributed windowed top-k frequent items problem consists in returning the top-k most frequent items received among all $\mathcal{N}$ nodes from time $t - W$ to time t.*

In the following we denote our algorithm by FTK$_{CE}$ and the one proposed by Song et al. [7] by FTK.

IV. Algorithms Description

A. Preliminaries

At the core of FTK and FTK$_{CE}$ algorithms lies a very simple yet powerful mechanism: coin flipping. Tossing a coin can be viewed as a Bernoulli process or a sequence of independent and identically distributed Bernoulli trials. Each trial has exactly two outcomes (e.g., *0* and *1* or *tail* and *head*) and the same probability p to obtain 1 or head (H).

Definition 4 (Bernoulli process)**.** *A Bernoulli process is a finite or infinite sequence of independent random variables $X_1, X_2, X_3, \ldots$, such that:*

- *For each integer i, the value of X_i is either 0 or 1;*
- *For all values of i, the probability that $X_i = 1$ is the same number p.*

Frequency correlated stochastic value. The Bernoulli process we consider here is the number of trials needed to obtain 0 (i.e., first hit time), which precisely corresponds to tossing a coin until the first tail occurs (see Figure 2). Clearly, more attempts implies longer hitting times. Thus the first hit time is a stochastic value statistically correlated to the number of attempts.

To get an insight on the correlation between the maximum hitting time and the number of trials, consider the experiment of obtaining two consecutive heads (until the first tail) with respectively one and two trials of a fair coin (i.e., $p = 1/2$). As shown in Figure 3, there is only one way to obtain "HHT" with one trial, while there are three possibilities with two trials.

Overview. The left heatmap in Figure 4 presents the global pattern of the hitting time probability evolution as a function of the number of trials. As can be seen, there is a logarithmic relationship between the number of trials and the maximum expected number of consecutive heads. For more empirical and theoretical evidence of this logarithmic behaviour, one can consult for example [16], [17], [18]. A logarithmic relationship implies that for a high number of trials, the correlation with the maximum hitting time is less significant than for a small number of trials. On the other hand, there is a linear relationship between the number of trials and

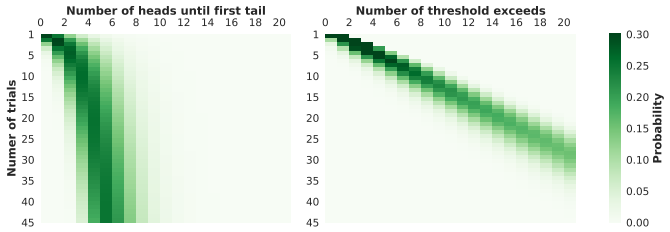


Fig. 4. Heatmap of probabilities to obtain a given number of consecutive heads (left) or a given number of threshold exceeds (right) as a function of the number of trials.

the number of times the hitting time exceeds a given threshold (see right heatmap in Figure 4).

B. Original approach - FTK

The original approach to identify the top-k most frequent items is to retain for each item in the stream the maximum number of consecutive heads obtained over all the occurrences of this item at each time unit. For each time unit, only the k items with the highest maximum number of consecutive heads are retained. Finally, the top-k most frequent items over a sliding window of W time units are the items with the highest maximum number of consecutive heads among all the items retained during the W time units.

It turns out that this technique is efficient for very skewed data distributions, but it is no more the case in presence of more homogeneous data distributions where items have comparable frequencies. This is in part due to the logarithmic relationship between the number of trials and the expected maximum number of consecutive heads as observed in Figure 4. In other words, the condition for FTK to correctly detect the k most frequent items is a large frequency gap between the top-k most frequent items and the other ones. Moreover, this gap should increase as a function of the stream length.

C. Our approach - FTK$_{CE}$

To address this important limitation, we propose another stochastic value, which is robustly correlated to the item's frequency, while keeping as much as possible the properties of the initial approach as detailed in the following.

Instead of keeping only the maximum number of consecutive heads, our algorithm counts the number of times the Bernoulli process (i.e., number of consecutive heads) of an item has exceeded a dynamically defined threshold θ. Relying on a threshold, can be viewed as filtering out items with low frequencies, which resembles in some aspects to the mixed techniques presented in Section II-C. As will be shown in the sequel, the value of this threshold has an impact on both the precision of the returned top-k items and on the space complexity of the algorithm.

As previously described, our algorithm FTK$_{CE}$ is locally run at nodes $S_1, \ldots, S_{\mathcal{N}}$, and at the specific coordinator node. The pseudo-code run at a given node S_i, associated with its stream σ, is presented in Algorithm 1. The main data structure maintained by Algorithm 1 is a list denoted by Γ. Each element

Algorithm 1: MAINTAINFTK$_{CE}$ (σ, θ)

Input : σ: data stream,
θ: hitting time threshold
Output: continuously evolving data structure Γ

```
foreach time unit t do
    Γ ← empty list
    foreach item u that arrives at t do
        l ← BERNOULLIPROCESS(p)
        if l ≥ θ then
            if u ∈ Γ then
                (u, c_u) ← (u, c_u + 1)
            else
                Γ ← Γ ∪ (u, 1)
    Send Γ to the coordinator
```

of Γ is a tuple (u, c_u), where c_u represents the number of times the Bernoulli process of an item u has exceeded a given threshold θ. Once all items received during a time unit t have been processed, node S_i sends to the coordinator its list Γ.

The pseudo-code run by the coordinator node is described in Algorithm 2. The coordinator maintains a first-in first-out linked list denoted by Γ_{coord}. At each time unit, the coordinator merges the information sent by $S_1, \ldots, S_{\mathcal{N}}$ to extract the top-k most frequent items received during the last W time units over the $\mathcal{N}$ distributed data streams. Technically, at each time unit and for each item u in $(u, -)$ of Γ_i, the coordinator sums the total number of times item u has exceeded threshold θ across all the nodes S_i and stores this information in $\Gamma_{coord}[0]$. Upon query the coordinator sums all c_u^{coord} of each item u in $(u, -)$ of Γ_{coord} over the last w time units. Let c_u^Q be this sum. The coordinator returns then the top-κ items u with the highest c_u^Q values (see Section IV-D, flexibility of parameters).

D. Properties of FTK and FTK$_{CE}$

In this section we present the four main properties common to both approaches, which are essential to answer the distributed and windowed top-k most frequent items problem. These properties are a direct consequence of the independence of the Bernoulli processes.

Sliding window. The first and most useful property of these algorithms is their memoryless feature, i.e., every time an item is received both algorithms attribute a stochastic value to this item which is independent from what happened in the past. This is the feature which allows us to deal with sliding windows.

Distributed monitoring. The above feature is also the one which allows us to deal with distributed streams. Thus, a distributed execution of both algorithms produce exactly the same result at the coordinator node as if all $\mathcal{N}$ data streams were directly received by a single node (e.g., the coordinator).

Algorithm 2: RETRIEVETOPK ($\Gamma_1, \ldots, \Gamma_{\mathcal{N}}$, $w = W$, $\kappa = k$)

Input : $\Gamma_1, \ldots, \Gamma_{\mathcal{N}}$: local nodes updates,
w: query size of the sliding window (default $w = W$),
κ: query number of top items (default $\kappa = k$)

Output: the top-κ most frequent items during the last w time units

```
foreach time unit t do
    Remove last element Γ_coord[W − 1]
    Append a new element at the beginning of Γ_coord
    Γ_coord[0] ← empty list
    for i ← 1 to 𝒩 do
        foreach (u, c_u) ∈ Γ_i do
            if u ∈ Γ_coord[0] then
                (u, c_u^coord) ← (u, c_u^coord + c_u)
            else
                Γ_coord[0] ← Γ_coord[0] ∪ (u, c_u)
Q ← empty list
for j ← 0 to w − 1 do
    foreach (u, c_u^coord) ∈ Γ_coord[j] do
        if u ∈ Q then
            (u, c_u^Q) ← (u, c_u^Q + c_u^coord)
        else
            Q ← Q ∪ (u, c_u^coord)
Sort Q in descending order by c_u^Q
TopItems ← first κ items of Q
return TopItems
```

Flexibility of parameters. An important property of both algorithms is to return the top-k most frequent items ordered according to their frequency. This allows both algorithms to dynamically answer any top-κ most frequent items query, with $\kappa \leq k$. In addition, the size of the sliding window can also be dynamically adjusted to any w, with $w \leq W$.

Permutation independence. Another important property of both approaches, is their resilience to any ordering manipulation of the items within any sliding window of the data streams.

All this properties shared by FTK and FTK$_{CE}$ are illustrated in Figure 5.

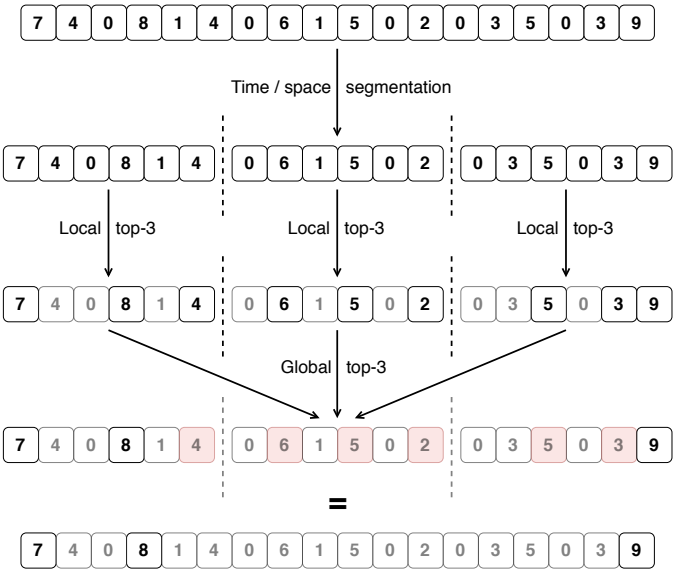


Fig. 5. Windowed and/or distributed FTK algorithm execution.

V. THRESHOLD DEFINITION

Even though both FTK and FTK$_{CE}$ algorithms use a Bernoulli process, they differentiate from each other by the way they aggregate the stochastic values. In the following of this section we detail the computation of the threshold θ used by our algorithm FTK$_{CE}$.

A. Rationale of the threshold θ in FTK$_{CE}$

Although the initial approach FTK is interesting when applied on skewed data distributions, simulations on homogeneous data distributions reveal its weakness. This is due to the choice of the aggregated value over the Bernoulli process. Indeed, the maximum number of heads is not a robust metric and it does not efficiently discriminate items based on their frequencies.

Example. As an illustrative example, think about identifying the top-1 most frequent item with FTK. Unless the most frequent item occurs more than 50% of times in the sliding window, the algorithm fails in average to correctly identify it as the top-1. Indeed, the top-1 expected precision corresponds to the proportion of its frequency it the sliding window, as the longest number of consecutive heads can be obtained by any other item due to the independence of the Bernoulli processes.

As previously said, what needs to be done is to compute an aggregated value which is more correlated to the item's frequency. Instead of keeping only the maximum number of consecutive heads, our algorithm counts the number of times the Bernoulli process of an item has exceeded a given threshold θ.

Such a threshold should be finely chosen to act like a filter. Indeed, a too restrictive filter (i.e., high θ) would only retain a subset of the true top-k items, while a too permissive one (i.e., low θ) would retain too many extra items. To fix a proper threshold, we combine two known theoretical results issued

from the coupon collector and the leader election problems. Briefly the coupon collector allows us to determine how many items should exceed the threshold to ensure that the true top-k most frequent items are among them. Then a result from the leader election problem allows us to derive which threshold enables this quantity of items to be retained.

B. Coupon collector problem

Suppose we have a set of N coupons (corresponding to the N different items in the universe) and we denote by p_u the probability of the coupon u to be drawn (according to the frequency distribution of items) such as $p_1+p_2+\cdots+p_N = 1$. For the sake of clarity and without any loss of generality, we suppose that coupons (and items) are ordered decreasingly according to their frequency, such as $p_1 \geq p_2 \geq \cdots \geq p_N$.

The coupon collector problem consists in determining the distribution of the number T_N of coupons that need to be drawn with replacement from the set $\{1, 2, \ldots, N\}$ in order to obtain the full collection of N different coupons.

The generalized version of the coupon collector problem allows us to consider a collection of coupons $\{1, 2, \ldots, x\}$ where $\mathbf{p} = (p_1, p_2, \ldots, p_x)$ is not necessarily a probability distribution, i.e., $p_1 + p_2 + \cdots + p_x \leq 1$. This models the presence of a null coupon 0, which is drawn with probability $\mathbf{p_0} = 1 - (p_1 + \cdots + p_x)$ and which is not allowed to belong to the considered collection.

In our case, we consider the collection formed by the k most frequent coupons $\{1, 2, \ldots, k\}$ and the null coupon 0 is formed by all other coupons $\{k+1, k+2, \ldots, N\}$ which is drawn with probability $\mathbf{p_0} = 1 - (p_1 + \cdots + p_k)$.

From [11] we have:

$$\mathbb{E}[T_k(\mathbf{p})] = \sum_{i=0}^{k-1} (-1)^{k-1-i} \sum_{|J|=i} \frac{1}{1 - (\mathbf{p_0} + P_j)}, \qquad (1)$$

where $P_j = \sum_{j \in J} p_j$ and J is a subset of $\{1, 2, \ldots, k\}$ of size i. $\mathbb{E}[T_k(\mathbf{p})]$ represents the minimum expected number of coupons to be drawn in order to obtain the collection formed by the k most frequent coupons. And once we have the number of all coupons (null and from the collection) to be drawn, we can use the next result from the leader election problem to compute the θ threshold.

C. Leader election problem

The leader election problem is fundamental in distributed systems. A common use case is the selection of a coordinator during the initialization of a service or after the failure of an existing one. The most studied issues in the leader election problems are their time and communication complexities which are essential to adapt to large scale systems. In this paper we are interested in the behaviour of the number of contestants during the election rounds. This is related to space complexity, since knowing in advance the distribution of survivors throughout the election process can lead to a better management of resources. We consider here a randomized election algorithm with the binomial splitting protocol and we adopt the definition of survivors from [19].

Assume there are n contestants (corresponding to the n items in the data stream) and every one flips a coin, not necessarily fair, but common to all contestants. We note p the probability of obtaining head and $1-p$ the one of obtaining tail. Contestants who obtain head are allowed to compete for the next round, while those who obtain tail are eliminated from the election. The set of contestants allowed to compete for the next round is called the advancing set and is denoted by K_n. In the coin flipping case, $K_n \sim Bin(n, p)$, hence the name of binomial splitting protocol. K_n can be seen as the sum of n Bernoulli random variables, with probability p to get 1 and $1-p$ to get 0. The number of contestants remaining after t rounds are called survivors and are designated by $\tilde{S}_{n,t}$. Note that as we are not directly interested in electing a winner, we consider the winnerless process where the algorithm continues even if there remains only one contestant. We are precisely interested in the number of survivors after a given number of election rounds. The distribution of the random variable $\tilde{S}_{n,t}$ is given by Kalpathy, Mahmoud and Rosenkrantz [19].

Theorem 1 (**Survivors distribution** [19])**.** *Suppose we conduct a leader election among n contestants, in which a fair selection of a subset of contestants of a random size $K_n = Bin(n, p)$ advance to the next round, and the algorithm is applied recursively on that subset, till all contestants are eliminated (exactly as in the elimination by the coin flipping process). The number of survivors, $\tilde{S}_{n,t}$, has the binomial distribution of $Bin(n, p^t)$.*

Proof. For self-contained reasons, we include the proof of Kalpathy et al [19]. The proof is done by induction on t that $\phi_{\tilde{S}_{n,t}}(x)$, the moment generating function of $\tilde{S}_{n,t}$, is $(1 - p^t + p^t e^x)^n$, which is that of $Bin(n, p^t)$. At $t = 0$, $\tilde{S}_{n,0} = n = Bin(n, p^0)$, providing a basis for the induction. Suppose now that, for $t \geq 1$, $\tilde{S}_{n,t-1}$ is distributed like $Bin(n, p^{t-1})$. If exactly k contestants survive till round $t-1$, of these $Bin(k, p)$ will advance to compete in round t. Thus, letting $q = 1 - p$, we have the conditional expectation

$$\mathbb{E}\Big[e^{x\tilde{S}_{n,t}} \mid \tilde{S}_{n,t-1} = k\Big] = \sum_{j=0}^{k} \binom{k}{j} p^j q^{k-j} e^{xj} = (q + pe^x)^k.$$

So, we have the unconditional expectation

$$\phi_{\tilde{S}_{n,t}}(x) = \sum_{k=0}^{n} (q + pe^x)^k \Pr[\tilde{S}_{n,t-1} = k],$$

which gives (by the induction hypothesis)

$$\begin{aligned}\phi_{\tilde{S}_{n,t}}(x) &= \sum_{k=0}^{n} (q + pe^x)^k \binom{n}{k} (p^{t-1})^k (1 - p^{t-1})^{n-k} \\ &= ((q + pe^x)p^{t-1} + 1 - p^{t-1})^n \\ &= (1 - p^t + p^t e^x)^n,\end{aligned}$$

completing the induction. □

D. Threshold θ computation

Technically, threshold θ is computed as follows. We use Formula (1) to determine the expected minimum number of items $\mathbb{E}[T_k(\mathbf{p})]$ to be drawn with replacement and according to their frequency distribution in the data stream in order to ensure that in expectation the true top-k most frequent items are among them. Then we determine the number of election rounds t to be performed in order to elect in expectation the number of items previously defined. This number of rounds corresponds to the value of t when $\mathbb{E}[\tilde{S}_{n,t}] = \mathbb{E}[T_k(\mathbf{p})]$, i.e., when the number of survivors $\tilde{S}_{n,t}$ equals in expectation to the minimum number of items to be drawn containing the true top-k $T_k(\mathbf{p})$.

In the case of a binomial splitting protocol, $\tilde{S}_{n,t} \sim Bin(n, p^t)$ (cf. Theorem 1). Thus, t must be such that $\mathbb{E}[Bin(n, p^t)] = \mathbb{E}[T_k(\mathbf{p})]$ and since $\mathbb{E}[Bin(n, p^t)] = np^t$, we have:

$$t = \frac{\log(\mathbb{E}[T_k(\mathbf{p})]) - \log(n)}{\log(p)}$$

Finally, threshold θ is defined as $\theta = t$.

In practice, as the data stream length and the frequencies of the top-k most frequent items are unknown, the computation of t is impossible. Nevertheless, the threshold θ can be heuristically initialized and dynamically maintained during the execution of FTK$_{CE}$. In Section VI we consider $\theta = \lceil t \rceil$, since the ceil value of t maximizes the error of FTK$_{CE}$.

VI. Performance evaluation

The present section contains results on simulated data and the main aspect covered here is the precision/recall of our approach (FTK$_{CE}$) compared to the original one (FTK) proposed by Song et al. in [7], which is so far, and to the best of our knowledge, the most impressive solution to solve the (distributed) windowed top-k problem.

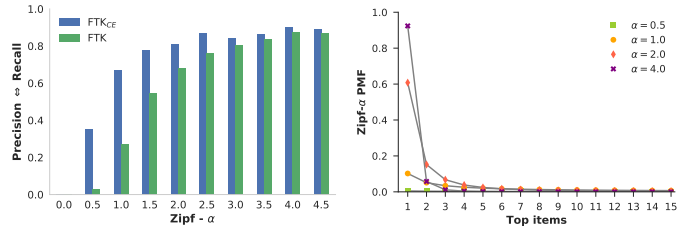
By precision, we mean the number of top-x items, with $1 \leq x \leq k$, correctly detected by the algorithms divided by the total number of detected items, i.e., k by design. A particularity of the top-k problem is that precision and recall are equivalent in this case. Indeed, the number of false positives in the returned top-k set corresponds exactly to the number of false negatives not returned by the algorithm.

A. Simulation protocol

To analyze the performance of our approach, we consider simulated data following zipfian distributions, as the family of power law distributions are widely observed in real word data sets and natural phenomena [20], [21].

Furthermore, all of the following experiments have been realized during an unique sliding window on a single node. Since in accordance with the proprieties described in Section IV-D, both algorithms will output the same precision for any sliding window over multiple nodes as long as the data streams keep the same distribution shape.

Concretely, we generate $n = 10^6$ items containing up to $N = 10^4$ distinct items and whose frequencies follow a Zipf-α distribution with $0 \leq \alpha \leq 4.5$. Note that Zipf-0 corresponds to the uniform data distribution and Zipf-4.5 corresponds to a extremely skewed data distribution (see Figure 6b). Moreover, the results presented below are an average of 10^3 algorithms executions and we also fix by default $k = 15$ for clarity and readability of graphs.



(a) (b)

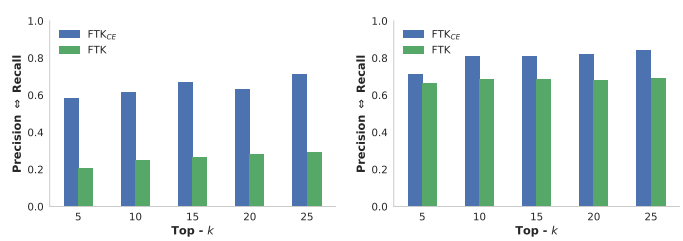
Fig. 6. FTK$_{CE}$ and FTK top-15 precision (a) and probability mass function of the first 15 most frequent items (b) for different Zipf distributions.

B. Items frequency distribution

Figure 6a compares the precision of both FTK$_{CE}$ and FTK algorithms as a function of different Zipf-α items frequency distributions, when queried to answer the top-15 most frequent items problem. FTK$_{CE}$ is capable to detect frequent items even for relatively flat Zipf-1 distributions (see Figure 6b). FTK starts to perform equivalently only from the Zipf-2 items frequency distribution, in which case just the top-1 most frequent item occurs more than 60% of times.

C. Number of top items

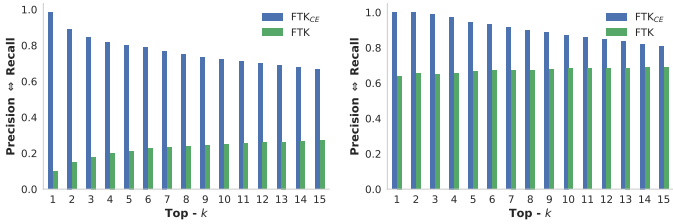
Both solutions perform consistently when employed to answer top-k frequent items queries for different values of k (see Figure 7). Moreover, the precision of FTK$_{CE}$ is improving when k is increasing for both frequency distributions of items, while for FTK it is only the case for the Zipf-1 distribution.



(a) Zipf-1 (b) Zipf-2

Fig. 7. Top-k precision, with $k \in \{5, 10, 15, 20, 25\}$.

As seen in Section IV-D, a common property of both FTK$_{CE}$ and FTK algorithms are their ability to answer any top-κ query as long as $\kappa \leq k$. Figure 8 compares their precision when asked for all possible top-κ queries, i.e., $1 \leq \kappa \leq k = 15$, which confirms the impressive behavior of our algorithm for all intermediate top-κ queries.



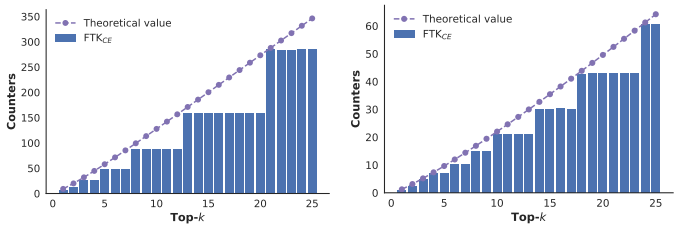
(a) Zipf-1 (b) Zipf-2

Fig. 8. Intermediate top-κ query precision, with $1 \leq \kappa \leq k = 15$.

D. Space complexity

In order to obtain an empirical overview of the memory cost, Figure 9 presents the number of counters that is necessary for the FTK$_{CE}$ algorithm in order to answer different top-k queries, for $1 \leq k \leq 25$. Simulation results show that the good precision of the FTK$_{CE}$ algorithm for a relatively homogeneous Zipf-1 distribution comes at a cost in terms of required memory. Nevertheless, in order to answer the top-25 most frequent items query, FTK$_{CE}$ uses less than 300 counters, i.e., less than 3 % compared to the total possible number of different items $N = 10000$.

Note that the different "steps" are due to the different values of threshold θ. For information, we plot the theoretical number of counters that would be required if the thresholds were directly defined as $\theta = t$, i.e., a float value (cf. Section V-D).



(a) Zipf-1 (b) Zipf-2

Fig. 9. Number of counters used by the FTK$_{CE}$ algorithm to answer the top-k most frequent items problem, for $1 \leq k \leq 25$.

VII. Conclusion

In this paper we have presented a new approach derived from the state-of-the-art solution provided by Song et al. [7] in order to answer the distributed sliding window top-k most frequent items problem. We have shown some important improvements of the precision of the returned top-k items by our method, especially for homogeneous frequency distributions of data items.

In order to mitigate the necessity to estimate the probability distribution vector $\mathbf{p}$ of the true top-k most frequent items, we are currently investigating the means to incorporate or bypass this limitation, although, it can already be done using techniques presented in Section II. We plan to analyze more in depth the theoretical behavior of our algorithm, such as (ϵ, δ)-error approximation or memory and communication cost bounds.

References

[1] E. D. Demaine, A. López-Ortiz, and J. I. Munro, "Frequency Estimation of Internet Packet Streams with Limited Space," in *Proceedings of the 10th Annual European Symposium on Algorithms*, 2002, pp. 348–360.

[2] G. Cormode and S. Muthukrishnan, "An improved data stream summary: the count-min sketch and its applications," *Journal of Algorithms*, vol. 55, no. 1, pp. 58–75, apr 2005.

[3] G. Cormode and M. Hadjieleftheriou, "Methods for finding frequent items in data streams," *The VLDB Journal*, vol. 19, no. 1, pp. 3–20, feb 2010.

[4] Z. Wei, G. Luo, K. Yi, X. Du, and J.-R. Wen, "Persistent Data Sketching," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data - SIGMOD '15*, 2015, pp. 795–810.

[5] N. Homem and J. P. Carvalho, "Finding top-k elements in a time-sliding window," *Evolving Systems*, vol. 2, no. 1, pp. 51–70, mar 2011.

[6] O. Papapetrou, M. Garofalakis, and A. Deligiannakis, "Sketching distributed sliding-window data streams," *The VLDB Journal*, vol. 24, no. 3, pp. 345–368, jun 2015.

[7] C. Song, X. Liu, and T. Ge, "Top-k Frequent Items and Item Frequency Tracking over Sliding Windows of Any Sizes," in *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, apr 2017, pp. 199–202.

[8] J. Misra and D. Gries, "Finding repeated elements," *Science of Computer Programming*, vol. 2, no. 2, pp. 143–152, nov 1982.

[9] N. Alon, Y. Matias, and M. Szegedy, "The space complexity of approximating the frequency moments," in *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing - STOC '96*, 1996, pp. 20–29.

[10] A. Metwally, D. Agrawal, and A. El Abbadi, "Efficient Computation of Frequent and Top-k Elements in Data Streams," in *Proceedings of the 10th international conference on Database Theory*, 2004, pp. 398–412.

[11] E. Anceaume, Y. Busnel, N. Rivetti, and B. Sericola, "Identifying Global Icebergs in Distributed Streams," in *2015 IEEE 34th Symposium on Reliable Distributed Systems (SRDS)*, sep 2015, pp. 266–275.

[12] M. Cafaro, M. Pulimeno, I. Epicoco, and G. Aloisio, "Mining frequent items in the time fading model," *Information Sciences*, vol. 370-371, pp. 221–238, nov 2016.

[13] H. Tang, Y. Wu, T. Li, C. Han, J. Ge, and X. Zhao, "Efficient Identification of TOP-K Heavy Hitters over Sliding Windows," *Mobile Networks and Applications*, pp. 1–10, may 2018.

[14] R. Keralapura, G. Cormode, and J. Ramamirtham, "Communication-efficient distributed monitoring of thresholded counts," in *Proceedings of the 2006 ACM SIGMOD international conference on Management of data - SIGMOD '06*, 2006, p. 289.

[15] R. Ben-Basat, G. Einziger, R. Friedman, and Y. Kassner, "Heavy hitters in streams and sliding windows," in *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, apr 2016, pp. 1–9.

[16] J. Kinney, "Tossing Coins Until All Are Heads," *Mathematics Magazine*, vol. 51, no. 3, pp. 184–186, 1978.

[17] M. F. Schilling, "The Longest Run of Heads," *College Mathematics Journal*, vol. 21, no. 3, pp. 196–207, 1990.

[18] J. Túri, "Limit theorems for the longest run," *Annales Mathematicae et Informaticae*, vol. 36, pp. 133–141, 2009.

[19] R. Kalpathy, H. M. Mahmoud, and W. Rosenkrantz, "Survivors in leader election algorithms," *Statistics & Probability Letters*, vol. 83, no. 12, pp. 2743–2749, dec 2013.

[20] L. A. Adamic and B. A. Huberman, "Zipf's law and the Internet," *Glottometrics 3*, pp. 143–150, 2002.

[21] M. Newman, "Power laws, Pareto distributions and Zipf's law," *Contemporary Physics*, vol. 46, no. 5, pp. 323–351, sep 2005.