TOPiCo: Detecting Most Frequent Items from Multiple High-Rate Event Streams

Valerio Schiavoni[*], Etienne Rivière, Pierre Sutra, Pascal Felber
Université de Neuchâtel, Switzerland

Miguel Matos, Rui Oliveira
INESC TEC & University of Minho, Portugal

ABSTRACT

Systems such as social networks, search engines or trading platforms operate geographically distant sites that continuously generate streams of events at high-rate. Such events can be access logs to web servers, feeds of messages from participants of a social network, or financial data, among others. The ability to timely detect trends and popularity variations is of paramount importance in such systems. In particular, determining what are the most popular events across all sites allows to capture the most relevant information in near real-time and quickly adapt the system to the load. This paper presents TOPiCo, a protocol that computes the most popular events across geo-distributed sites in a low cost, bandwidth-efficient and timely manner. TOPiCo starts by building the set of most popular events locally at each site. Then, it disseminates only events that have a chance to be among the most popular ones across all sites, significantly reducing the required bandwidth. We give a correctness proof of our algorithm and evaluate TOPiCo using a real-world trace of more than 240 million events spread across 32 sites. Our empirical results shows that (i) TOPiCo is timely and cost-efficient for detecting popular events in a large-scale setting, (ii) it adapts dynamically to the distribution of the events, and (iii) our protocol is particularly efficient for skewed distributions.

Keywords

top-k query, top-k frequency, distributed systems, event processing

Categories and Subject Descriptors

C.2.4 [**Computer-Communication networks**]: Distributed systems

[*]Corresponding author: valerio.schiavoni@unine.ch

1. INTRODUCTION

The collection of aggregates and statistical metrics over online data streams has attracted considerable attention from both academia and industry over the past decade. Mining the properties of such data streams can be used in various contexts, ranging from targeted advertisement [6], network analysis [7, 25], or automated virus detection [20]. Of the various statistical information that can be computed over a stream, identifying the distribution of the events is of practical interest. This information might help a content delivery network infrastructure to dimension its caches using the stream of web access logs. It can also detect sudden changes in popularity, e.g., flash crowds phenomena or denial of service attacks.

Practical streaming systems exhibit heavy-tailed distributions. Moreover, in a vast majority of use cases, only the most frequent items are required. As a consequence, collecting and storing the frequency of all events to compute only the k most frequent ones is a waste of resources. This observation is particularly acute in a geo-distributed setting, where distinct sites might receive distinct stream of events. Detecting the k most frequent items is a form of top-k query processing, where the query is simply the sum of occurrences grouped by item. As a consequence, we shall use the term *top-k frequent items* in the remainder of this paper.[1]

We are interested in the construction of the top-k frequent items from the union of multiple streams. These different streams are generated at multiple geographically distant locations. We consider the following motivating scenario. A set of geo-distributed servers located in different regions of the world support the information needed for a large-scale event, e.g., the Olympics or the FIFA World Cup. Servers receive and emit messages, similarly to the Twitter service, for the local area. Participants and spectators can comment and react while the events occur, online. Messages are tagged with keywords regarding the event, e.g., the names or moods of the participants. Each site maintains the top-k frequent keywords over a sliding window, allowing to observe the trends for a particular region. We are also interested in computing the most frequent keywords at the scale of the geo-distributed infrastructure. Such information is used

[1]As pointed out by [26], our problem differs from the simple top-k problem where unique items with the highest values for a given attribute are returned, which is merely a selection problem [2]. A top-k *query* problem on the opposite, e.g., top-k counting and top-k frequent items, requires to aggregate multiple instances of the same item from different sites.

locally, for instance, to see in near real-time the differences between local and global trends. A naive solution for such a scenario consists in redirecting all the streams towards all the sites. Obviously, this solution does not scale with the number of sites, nor with the geo-distribution. Besides, although solutions exist for speeding up data streams over multiple data centers [22], they have huge bandwidth costs, as well as higher-than-required computational power.

We are thus interested in maintaining the *global* top-k frequent items in a multi-site information system. The global top-k frequent items set is referred as $\mathcal{G}$ in the remainder of this paper. We assume that each site maintains, using an existing single-site algorithm, the *local* top-k frequent items for its own stream. This local top-k frequent items set for site i is denoted as $\mathcal{L}_i$. Our goal is to construct, at each site i, a version of the global top-k frequent items, denoted as $\mathcal{G}_i$. The construction of $\mathcal{G}_i$ is based on a combination of $\mathcal{L}_i$ and of information received from the other sites. The global top-k frequent items set $\mathcal{G}$ is never materialized as there is no centralized entity receiving all the streams without transmission delays. The objectives for the construction of $\mathcal{G}_i$ at each site are twofold. First, we want to minimize the amount of information that is exchanged between sites. Second, we want to minimize the deviation between $\mathcal{G}_i$ maintained at one site, and the hypothetical and ideal content of $\mathcal{G}$, as constructed by an omniscient observer collecting all the streams in real time.

Contributions. We make the following contributions to the top-k frequent items problem. We propose TOPiCo, a novel protocol for computing an accurate view of the globally most frequent items at the scale of multiple geo-distributed sites. Our protocol leverages the presence of a continuously maintained local list of the top-k frequent items at each site. Sites exchange such lists up to a certain depth, while ensuring that the global view $\mathcal{G}_i$ computed at each site effectively contains the appropriate elements and their respective frequencies, within reasonable and practical delay. We provide a correctness proof of the protocol, as well as an extensive experimental evaluation using a prototype and fed with traces of 240 million events received on 32 sites, collected during the FIFA World Cup'98 [1]. The evaluation confirms that TOPiCo is a lightweight approach and shows that it is able to dynamically adapt to the items distribution. The rest of the paper is organized as follows: Section 2 discusses related work. Section 3 precisely formulates the top-k frequent items problem and circumscribe the conditions under which this problem is solvable. Section 4 details our TOPiCo protocol and covers its correctness proof. Section 5 presents our experimental evaluation of the TOPiCo prototype. Finally, Section 6 concludes the paper.

2. RELATED WORK

The problem of efficiently computing the results of top-k frequent items from a stream has been considered both in centralized and distributed settings. Our focus is on the construction of the *global* top-k frequent items sets $\mathcal{G}_i$ at each sites of a distributed system rather than the construction of the *local* top-k frequent items sets $\mathcal{L}_i$ for each individual sites. The construction of $\mathcal{L}_i$ employs a centralized algorithm, which can be implemented using a stream processing engine [3, 12]. As this construction is not the focus in this paper, we use a simple counting-based approach and concentrate on the distributed aspects. We provide nonetheless a review of centralized solutions and alternatives that can be used to build $\mathcal{L}_i$ at each site, as well as similar and related problems that could benefit from our distributed algorithm. Then, we review distributed algorithms and protocols allowing to compute $\mathcal{G}$, by collecting information from multiple sites. Note that the problem of top-k frequent items is sometimes called the *heavy hitters* problem in the literature.

2.1 Centralized top-k frequent items

Ilyas et al. [13] present a survey of top-k query algorithms for centralized single site relational database systems. The queries that they consider include *counting* queries, which themselves include *frequent items* queries. The survey however does not consider the data streaming model but only instantaneous queries.

The FREQUENT algorithm of Misra and Gries [18] was an early approach proposed for the detection of frequently occurring items in an infinite stream. It allows outputting the set of elements that account for more than a fraction $\frac{1}{f}$ of the total stream size, i.e., to return the set $j : c_j > \frac{c}{f}$ where c is the size of the stream seen so far, and c_j is the number of occurrences of element j. This is a different problem than ours, as top-k frequent items may be less present in the stream than $\frac{c}{f}$ even for $f \geq k + 1$. The algorithm maintains $f - 1$ counters, associated with unique items. For each item seen in the stream, the corresponding counter is incremented if it exists (or created if there are unused counter). Otherwise, all counters are decremented and freed when they reach zero. The counters are eventually associated with up to $f - 1$ items that are present in more than a fraction $\frac{1}{f}$ of the stream.

The *Count-Min* sketch of Cormode and Muthukrishnan [5] is a data structure dedicated to the summarization of data streams. It can allow detecting the most frequent items, among several other operations. It would be a possible alternative for the computation of the $\mathcal{L}_i$ at each site.

Lahiri et al. [14] consider the problem of tracking persistent items in a stream. This is a different, but complementary problem to top-k frequent items. Computing both sets can allow indicating trends over time or detect sudden changes in popularity of items.

Wang et al. [24] propose a framework to execute top-k *pattern* queries. Such queries allow recognizing the most frequent sequences of events for which the patterns corresponding to the interdependency conditions apply. This technique uses adaptive join scheduling strategies and stratified stream graphs. The output could be used as the local top-k frequent items set $\mathcal{L}_i$ used in our algorithm, allowing to compute the result of a global top-k *pattern* query.

Wong and Fu [26] present a probabilistic solution for top-k frequent *item sets* over a stream. The problem they consider is more general than the one we consider in this paper. Their goal is to detect the most frequent *item sets*, i.e., set of l items that frequently appear together from a stream of transactions. Note that the two problems are equivalent if $l = 1$, i.e., when considering individual items. The authors propose two algorithms that can derive the top-k frequent *item sets* from a complete stream without prior knowledge of its statistical characteristics: an algorithm based on Chernoff bounds and the Top-k *Lossy Counting* algorithm. These algorithms could also be candidate for computing $\mathcal{L}_i$ at each of site. Furthermore, when combined with our contribution, they allow to compute the global top-k frequent *item sets* at each site.

The Vitter's reservoir [23] samples uniformly a complete stream using a fixed-size reservoir, which is simply an array of samples. The reservoir allows estimating the distribution of events from the beginning of the stream. If this distribution of events presents enough skew (e.g., the popularity follows a power law), a large enough reservoir may allow answering top-k frequent items. However, there is no guarantee: Items belonging to the most frequent ones may simply be missed if there are not sampled enough in the reservoir.

2.2 Distributed top-k frequent items

The *threshold algorithm* (TA) of Fagin et al. [9] computes the top-k frequent items from a collection of items in independent sets. The algorithm does not consider data streams but static sets, which correspond to independent disk drives in a relational database setting. They can be considered as sites in our system model. A coordinator process computes $\mathcal{G}$ through several rounds of interaction with the sites. Each of the site maintains its list of items in decreasing frequency order. The algorithm goes down the list at all sites in parallel. In its TA-random variation, the algorithm computes for each new item its aggregate value. This requires being able to perform random accesses to the lists. A variant for systems, where only sequential access is possible (or preferred), is named the TA-sorted or NRA in [9], as well as Stream-Combine in an independent work by Guntzer et al. [11]. TA-sorted first computes the set of items that are part of the top-k frequent items, without calculating their exact aggregate count values. It requires an additional final pass to perform these aggregations. For both TA-random and TA-sorted, thresholds are computed to allow determining when elements that are down the lists at each site will not be included in the top-k frequent items, that is, their aggregate value cannot be higher than the k^{th} element in $\mathcal{G}$. The threshold can be exact in the TA-random case or approximated by bounds of worst and best possible scores for TA-sorted. Since lists are sorted in decreasing frequency order, the algorithm can stop when new elements have frequency lower than the (worst case) threshold.

Both threshold algorithms require $O(N^2)$ operations, where N is the number of sites. The number of iterations is also unbounded, and depends on the similarity of rankings between the different $\mathcal{L}_i$ of the different sites . In a distributed setting, this prevents from giving any guarantee on the delay of calculation of $\mathcal{G}$ at the coordinator. While this might be acceptable in a relational database setting, it is not adapted in a distributed data streaming model. Such an arbitrary delay may incur an important and uncontrollable drift between the content of $\mathcal{G}_i$ at the coordinator and the actual $\mathcal{G}$ an omniscient observer would obtain. Furthermore, the number of iterations at a site depends on the popularity distribution of items at that particular site. This may lead to the content of $\mathcal{G}_i$ being based on sliding windows for uncorrelated time periods at the different sites. As a consequence, in both cases, the final content of $\mathcal{G}_i$ might be inadequate to effectively detect trends across all sites in a timely manner.

Probabilistic versions of the threshold algorithms named the Prob-sorted algorithms family, were proposed by Theobald et al. [21]. They produce estimations of $\mathcal{G}$ based on probabilistic score predictions. This allows reducing the number of accesses to the lists at each site, in particular when the order of elements highly differ from one site to another, but does not guarantee that the exact scores for the aggregate count values are computed.

The TPUT algorithm of Cao and Wang [4] addresses the limitation of the threshold algorithm for static data sets. It explicitly targets a distributed setting with a coordinator site that computes the value $\mathcal{G}$ through interaction with the other sites. Unlike the threshold algorithm, TPUT requires 3 rounds between the coordinator and the sites. In the first round, the algorithm computes a lower bound τ on the value associated with the k^{th} element in $\mathcal{G}$. Each site sends its top-k elements from $\mathcal{L}_i$ to the coordinator, which uses the bottom value computed by the aggregation of these lists as τ. In the second round, the coordinator selects the threshold $T = \frac{\tau}{N}$. All sites then send back the elements not previously sent with a frequency of at least T. At this point, the coordinator can determine an overset S of the elements that form $\mathcal{G}$. A third round is required to collect the actual values associated with the elements in S, in order to determine the final and exact content of $\mathcal{G}$.

TPUT operates on a static set; it does not consider the data streaming model, nor computations on sliding windows. Furthermore, it uses a single coordinator site, while our problem is to compute $\mathcal{G}_i$ at each site. TPUT could be instantiated with a coordinator at each site: In this case, each site would have to perform the three phases and pull information from all the other nodes. Our approach is the opposite. Each site decides on its own using only local information what it needs to send to the others. This allows a single, one way communication to inform about the evolution of $\mathcal{L}_i$, and therefore the changes to the $\mathcal{G}_i$ on other sites, in comparison with the three two ways communication and the resulting higher delays and load with TPUT. We also exploit the evolution of $\mathcal{L}_i$ through time whereas TPUT is essentially a one-shot algorithm that recomputes $\mathcal{G}$ from scratch for each new query.

Manjhi et al. [16] consider the top-k frequent items problem with multiple sites where sites are organized in a tree structure. Their approach is to build an approximate version of $\mathcal{G}$. They allow nodes to define the degree of precision, and introduce the notion of precision gradient that captures the precision of the combination of approximate frequency counts along the tree, with the goal of minimizing the bandwidth cost as much as possible. We do not consider the use of a rigid overlay between nodes, and we target a complete computation of the top-k frequent items at all nodes.

Michel et. al [17] present the KLEE system which target a range of top-k queries including counting, and thus cover top-k frequent items. This solution works for P2P networks, and it provides an approximate computation of $\mathcal{G}$. It is unclear how the solution can be evolved to support a data streaming model, or a model that supports dynamic data sets in general.

Some solutions are based on gossip-based protocols, where a periodic interaction takes place between pairs of nodes in a probabilistic manner, and with no complete membership information maintained at each node. Lahiri and Tirthapura [15] present such a gossip-based algorithm using adaptive sampling techniques. Their algorithm can consider absolute thresholds (more than a certain quantity of items are present in the system, regardless of its size), and relative thresholds similar to the model used by Misra-Gries [18]. The set of the *probabilistically most frequent items* is available at all peers after convergence. Sacha and Montresor [19] present another gossip-based protocol that targets the same problem but achieves a higher efficiency. Guerrieri et al. [10] present an evolution of the algorithm of Sacha and Montresor [19] to account for the addition and deletion of items. It is nonetheless unclear if any of these approaches could be adapted to sets of events that evolve very rapidly over time, and in particular to the data streaming model. Furthermore, even the addition and removal of items may be reflected after several rounds of gossiping as it is difficult to track and remove items that rapidly leave the window of interest. This indicates that such decentralized solutions are more adapted to static sets with complete periodic re-computation, or to slowly evolving sets where the delay of propagation is less an issue.

3. TOP-K FREQUENT ITEMS PROBLEM

This section defines the elements of our system model, then introduces the problem of computing the top-k frequent items over a distributed set of streams. Further, we state two results regarding the space complexity of every solution. These results serve as guidelines to our TOPiCo protocol.

3.1 System model

We consider a system composed of N distributed sites that communicate through message-passing. For the sake of simplicity, we assume that the communication graph is complete, and that sites are able to send/receive messages via a reliable communication medium. We shall precise our synchrony assumptions in the following.

Every site i receives a continuous stream s_i of events at some rate λ_i. Each event in the streams refers to some uniquely identified item (e.g., a topic, a keyword). We note *Items* the countable set of items, and we assume some ordering $<$ over *Items*. Every time a site i receives an event e, i tags e with the reception time. Based on this timestamping mechanism, every site i continuously keeps track of the events received within a time-based sliding window W_i of length τ.

The *top-k frequent items* problem requires to compute at each site a view of the most frequent items received globally, across all sites, within the last τ units of time. To model this problem, we consider that each site i holds two data structures $\mathcal{L}_i$ and $\mathcal{G}_i$, as described next:

- Variable $\mathcal{L}_i$ maintains the *local top-k frequent items*. This variables stores in order the items received at site i, together with their respective number of occurrences in the time window W_i (ties are broken according to $<$). For the sake of simplicity, we shall be assuming hereafter that $\mathcal{L}_i$ contains in fact *all* the items in W_i with their number of occurrences in a sound order.
- Variable $\mathcal{G}_i$ stores the *global top-k frequent items* as collected by site i. This means that $\mathcal{G}_i$ contains the local view at site i of the k most frequent items received globally in the streams $(\lambda_i)_i$ during the last τ units of time.

To illustrate the above data structures, consider that site i received items "x" at times 1 and 2, and respectively "y" and "z" at times 3 and 4. Further consider that the sliding window length equals 3. In such a case, at time $t = 3$, we have $\mathcal{L}_{i,3} = \{(\text{“x”}, 2), (\text{“y”}, 1)\}$, while at time $t = 4$, $\mathcal{L}_{i,4} = \{(\text{“x”}, 1), (\text{“y”}, 1), (\text{“z”}, 1)\}$ holds. Then, if we consider that $k = 1$ and $<$ is the Lexicographical order, we have $\mathcal{G}_{i,3} = \{(\text{“x”}, 2)\}$ and $\mathcal{L}_{i,4} = \{(\text{“z”}, 1)\}$.

The core task of any top-k frequent items protocol is to maintain at each site i a value of $\mathcal{G}_i$ consistent with the content of the streams $(s_i)_i$. Intuitively, we want to compare the local computation of $\mathcal{G}_i$ against an omniscient computation of the globally most frequent items. Next, we state such a notion in more formal terms.

3.2 Problem statement

Let us note $\mathcal{G}$ the top-k most frequent items in $(s_i)_i$ over the last period of τ units of time. In the top-k frequent items problem, the key information is the ordering of the items. We capture this by measuring the distance d between the ordering of the items in $\mathcal{G}$ and $\mathcal{G}_i$ at some site i.[2] For some site i, we shall note $d(\mathcal{G}_{i,t}, \mathcal{G}_t)$ the distance between $\mathcal{G}_i$ and $\mathcal{G}$ at time t.

DEFINITION 1. *A top-k frequency protocol is* perfect *when, for any ϵ, $lim_{t\to\infty} d(\mathcal{G}_{i,t}, \mathcal{G}_t) < \epsilon$ holds.*

Clearly, constructing such a perfect algorithm is not always possible. This might be for instance the case when the stream rate times the message delay between sites is higher than one. Indeed, whenever a site i received some information about the most frequent items at site j, such an information can be outdated by the arrival of a novel item at site j. The two results that follow further circumscribe the conditions under which a perfect solution is constructible for the top-k frequency problem. With more details, Lemma 1 proves that the problem requires a bound on the message delay between sites. Then, we show in Lemma 2 that the existence of a greatest element in *Items* for the order $<$ is necessary.

LEMMA 1. *No top-k frequent items protocol is perfect in an asynchronous distributed system.*

PROOF. (By contradiction.) Consider a system with two sites, i and j, and assume that $k = 1$ holds. In addition, suppose that x and y are the two sole items received respectively at sites i and j at rate $\frac{1}{\tau}$, starting from time 0. Furthermore, consider that $x > y$ holds for the arbitrary order defined to break ties. It follows that at any time t, x is the only item in $\mathcal{G}_t$. Since the system is asynchronous, there is no bound on the message delay between the two sites. In particular, for any value of τ, we might consider repeated arbitrary asynchrony periods longer than τ during which site j does not receive the messages from site i. During such a period, $\mathcal{G}_j$ cannot contain only item y as it would differ from $\mathcal{G}$. However, by a simple undistinguishability argument, we might also consider the exact same run up to that point, and consider now that site i does not received the events associated to x during the asynchrony period. Hence, during such a run, $\mathcal{G}_j$ should only contain y; a contradiction. $\square$

LEMMA 2. *Finding a perfect solution to the top-k frequent items problem requires a greatest element in Items for the order $<$.*

PROOF. (By contradiction.) Let us consider again two sites i and j and that $k = 1$. Stream s_j is empty, while stream s_i is a continuous sequence of distinct items $x_0, x_1, \ldots$, growing for the order $<$, and received at the fixed rate λ from

[2]Our problem definition does not depend on a specific distance function d. Meaningful distance functions include Levenshtein's or the Kendall-Tau rank distance [8]. We use Kendall-Tau in Section 5.

time 0. In addition, consider some non-null message delay from site i to site j. Clearly at time t, $\mathcal{G}_t$ equals $\{(x_{\lfloor \frac{t}{\lambda} \rfloor}, 1)\}$. On the other hand, observe that for any $k \geq 1$, at time $t = k \times \delta$, site j never received the item $x_{\lfloor \frac{t}{\lambda} \rfloor}$. Hence, the distance between $\mathcal{G}_{j,t}$ and $\mathcal{G}_t$ never converges toward 0. □

To accommodate with these results, we introduce two additional properties for top-k frequent items protocols.

Definition 2. *A protocol shall be* ϵ-good *when at all time t, for every site s_i, $d(\mathcal{G}_{i,t}, \mathcal{G}_t) < \epsilon$ holds.*

Definition 3. *A top-k frequent items protocol is* eventually perfect *when, $(\mathcal{G}_t)_t$ convergent implies $lim_{t \to \infty} d(\mathcal{G}_{i,t}, \mathcal{G}_t) = 0$.*

Section 5 shows that the TOPiCo protocol exhibits on average a 0.24-goodness factor for the Kendall tau rank distance during our experiments on the given dataset. Section 4.3 proves that TOPiCo is eventually perfect.

3.3 Resolvability

As pointed out previously, the goal of any top-k frequent items protocol is to exchange the minimal amount of information about the local top-k to ensure that $d(\mathcal{G}_{i,t}, \mathcal{G}_t)$ is minimal at each site. This means that a site should send only the items in $\mathcal{L}_i$ that are likely to enter in $\mathcal{G}$. Of course, at least the top-k items in $\mathcal{L}_i$ have to be sent to the other sites. However, it is also obvious that just sending only those entries is not enough to correctly compute $\mathcal{G}$. We state this simple observation in the lemma that follows.

Lemma 3. *Exchanging the top-k items from $\mathcal{L}_i$ among all sites is not sufficient to be eventually perfect.*

Proof. To prove the above claim, we exhibit a simple counter-example. We consider two sites i and j such that at some point in time we have: $\mathcal{L}_i = \{(a, 10), (x, 6)\}$ and $\mathcal{L}_j = \{(d, 9), (x, 5)\}$. In addition, let us consider that $k = 1$. If site i and j only exchange their first entries, i.e., respectively $(a, 10)$ and $(d, 9)$, we shall have at both sites $\mathcal{G}_i = \mathcal{G}_j = \{(a, 10)\}$. However, we clearly have that $\mathcal{G} = \{(x, 11)\}$. Hence, the previous approach never converges toward the correct solution. □

At a consequence of the previous result, we need to determine locally a value $l \geq k$ such that sending the top l items is sufficient to converge toward $\mathcal{G}$. Our next result shows that such a l exists by proving that a full information protocol is eventually perfect.

Lemma 4. *A full information protocol is an eventually perfect top-k frequent items protocol.*

Proof. Let us first recall that a full information protocol consists at each site in sending the local state every time this state changes and storing all historical data. Then consider some run of this protocol. Since $\mathcal{G}_t$ is convergent, $lim_{t \to \infty} \mathcal{G}_t$ exists. We note G such limit and T the time after which $\mathcal{G}_{t>T} = G$. Consider some tuple (x, ω) in G. From the definition of G, ω is the aggregated value of the number of occurrences of x in $(\mathcal{L}_{i,t})_i$ for every $t > T$. Hence after time T, computing the top-k frequent items on $(\mathcal{L}_{i,t})_i$ for any $t > T$ leads to G. Since we make use of a full information protocol, once every sites i broadcasts $\mathcal{L}_i$ after time t, we have eventually $\mathcal{G}_i = G$ at all sites. □

Despite a full information protocol is a correct solution, it requires to broadcast an information every time a novel event is received. This is not practical. On the contrary, the goal of our TOPiCo protocol is to allow sites constructing $\mathcal{G}$ efficiently, by forwarding as few information as possible. In the next section, we detail the internals of our approach, and prove that TOPiCo is eventually perfect.

4. THE TOPICO PROTOCOL

In this section, we describe the TOPiCo protocol in detail. We first start with an overview of our approach, while providing key insights on the internals of our solution. The concluding part of this section provides a formal proof that TOPiCo is eventually perfect.

4.1 Overview of the protocol

Each TOPiCo site i executes the following two tasks: (Update) Site i computes locally a list of *candidates* items that it broadcasts together with their local number of occurrences to all sites. (Disseminate) Upon receiving a list of candidates from some distant site j, a site i updates its global view of the most frequent items $\mathcal{G}_i$. To that end, i first sums-up for each item the contribution received in the candidate lists from the other sites (such contribution equals 0, if the item was not received). Then, site i sorts the global contributions and outputs $\mathcal{G}_i$. We consider that the system is synchronous, and that the update task occurs at frequency $1/\delta$.

TOPiCo constructs a list of candidates by determining at each site a value $l \geq k$ for which the top-l ranked items in $\mathcal{L}_i$ have a chance to enter in $\mathcal{G}$. Such an estimation is based on (i) the global number of occurrences of each item among the top-k in $\mathcal{G}_i$, (ii) the number of occurrences of each items in $\mathcal{L}_i$, and (iii) the candidates received by remote sites. In what follows, we cover with more details the internals of our approach, and how this estimation is computed.

4.2 TOPiCo in detail

Our first key observation in the design of TOPiCo is the following:

> (**Observation 1**) Consider an item x at some position lower than k in $\mathcal{G}$. If x enters in the top-k most ranked items in $\mathcal{G}$, then there exists a site i for which the number of occurrences of x at i times N is greater than the number of global occurrence of the item at position k in $\mathcal{G}$. Thus, item x is at site i a candidate to enter $\mathcal{G}$.

To actually transform the above observation into an algorithm, we must then accommodate with the fact that no site has access to $\mathcal{G}$. First, every site i executes the above *candidacy test* on the items in $\mathcal{L}_i$, using $\mathcal{G}_i$. Then, we make a second observation:

> (**Observation 2**) Consider that some item x passes the candidacy test at site i, i.e., denoting ω the number of occurrences of x in W_j, we have $\omega \times N > \mathcal{G}_i[k-1]$. Item x might fail the candidacy test at some other site j, for instance if j never receives x. Hence for every candidate it receives, site j must piggyback its local number of occurences.

We base our TOPiCo protocol on the above two observations. Algorithm 1 presents its pseudo-code. In addition

Algorithm 1: TOPiCo at process i

```
variables
    $candidates_i$ ;                                      // the candidates
    $\mathcal{L}_i$ ;                                      // local top-$k$
    $\mathcal{G}_i$ ;                                      // global top-$k$
    $\delta$ ;                                             // update period
task update
    upon receive $\langle UPDATE, C\rangle$ from $j$
        $candidates_i[j] \leftarrow C$
        foreach $(x, \_) \in C$ do
            $\mathcal{G}_i \leftarrow \mathcal{G}_i \setminus \{(x, \_)\}$
            $\Omega \leftarrow \sum_{(x,\omega)\in candidates_i} \omega$
            $\mathcal{G}_i \leftarrow \mathcal{G}_i \cup \{(x, \Omega)\}$
        $\mathcal{G}_i \leftarrow \{\mathcal{G}_i[0], \ldots, \mathcal{G}_i[l-1]\}$ ;   // keep top-$k$ items
task disseminate (every $\delta$ second)
    let $(\_, \Gamma) = \mathcal{G}_i[k-1]$ ;   // lowest score in top-$k$
    $l \leftarrow k$
    while $l < |\mathcal{L}_i|$ do
        let $(x, \omega) = \mathcal{L}_i[l]$
        if $\omega \times N \geq \Gamma$ then
            $l \leftarrow l + 1$ ;         // candidacy test passed
        else
            break
    $C \leftarrow \emptyset$ $n \leftarrow 0$
    while $n < l$ do
        $C \leftarrow C \cup \mathcal{L}_i[n]$
        $n \leftarrow n + 1$
    foreach $(x, \_) \in candidates_i$ do
        if $(x, \_) \notin C \wedge (x, \omega) \in \mathcal{L}_i$ then
            $C \leftarrow C \cup \{(x, \omega)\}$
    broadcast $\langle UPDATE, C\rangle$ to all sites
```

to $\mathcal{L}_i$ and $\mathcal{G}_i$, the protocol uses two additional local variables: δ defines the periodicity of the dissemination task, and *candidates*$_i$ is an array that contains for each site i, the last candidates received at site i from j.

In details, TOPiCo works as follows. The *update* task is in charge of maintaining the candidates at every site i. Upon the reception of a new set of candidates C from some site j (which might be i), the update task assigns C to *candidates*$_i[j]$ (line 8). Then, the computation of $\mathcal{G}_i$ takes place. For every item x in C, site i computes the aggregated value of x over all the candidates in *candidates*$_i$ and updates variable $\mathcal{G}_i$ accordingly (lines 10 to 12). Notice that at line 11, we write for simplicity $(x, \omega) \in candidates_i$ instead of considering (x, ω) in the multiset $\bigcup_j candidates_i[j]$. The update task ends by truncating $\mathcal{G}_i$ to only keep the first l entries (line 13).

The core routine of TOPiCo is the *disseminate* task. Its goal is to broadcast the candidates computed at site i with a periodicity of δ units of time. The candidates are the $l \geq k$ most frequent items in $\mathcal{L}_i$. To determine the value of l, the disseminate task first computes Γ, the number of occurrence of the last item in $\mathcal{G}_i$ (line 15). Then, it traverses $\mathcal{L}_i$ starting from position k (lines 17 to 22). Every time an item x successes the candidacy test, l is incremented (line 20); otherwise the loop ends (line 22). Site i collects the number of occurrences of items that successfully passed the candidacy test to form the candidates list C (lines 23 to 26). Then, site i appends to this list the candidates that were sent by other sites. (lines 27 to 29), and broadcasts the final content of C to all sites (line 30).

4.3 Proof of Correctness

This section is devoted to a formal proof of the correctness of TOPiCo. We formulate this result below then detail how to achieve it.

THEOREM 1. *The* TOPiCo *protocol is eventually perfect.*

PROOF. (By contradiction.) Let us consider some run ρ of the TOPiCo protocol. As $\mathcal{G}_t$ is convergent, we know that $lim_{t\to\infty}\mathcal{G}_t$ exists. Let G be that limit and T the time after which $\mathcal{G}_{t>T} = G$ holds. At some site j, we note $\mathcal{L}_j[l].\omega$ (respectively $\mathcal{G}_j[l].\omega$) the number of occurrences of the item at rank l in $\mathcal{L}_j$ (resp. $\mathcal{G}_j$), and for some item x, $\omega_{x,j}$ the number of occurrences of x in W. At every site j, recall that for every item z at position l in $\mathcal{L}_j$, we have $\mathcal{L}_j[l].\omega = \omega_{z,j}$.

Consider a time t after T in the run ρ. For the sake of contradiction, assume that an item x is in $\mathcal{G}$, but not in $\mathcal{G}_i$ at some site i. Name y the last item in $\mathcal{G}_i$. Since item y belongs to $\mathcal{G}_i$ and the system is synchronous, y also appears in every $\mathcal{G}_j$ at the same position $l_j \leq k$. Moreover, as item x appears in $\mathcal{G}$ and not y after time T. there must exist some site j_0 for which $\omega_{x,j_0} \times N > \mathcal{G}_{j_0}[l_j]$.

From the above analysis, it follows that for every item z higher than x (and including it) in $\mathcal{L}_{j_0}$. we have $\omega_{z,j_0} \times N > \mathcal{G}_{j_0}[k-1].\omega$, Hence, x passes the candidacy test (line 20) at site j_0. From which we deduce that x is among the candidates at that site, and is broadcast to all sites (lines 23 to 30). Then, every site receiving (x, ω_{x,j_0}), adds x to its candidates list, if it was not the case previously (lines 27 to 29). It follows that at the end of the computation round, x precedes y in $\mathcal{G}_i$. $\square$

5. EVALUATION

We evaluate TOPiCo using a real workload and a real implementation. The prototype is implemented using a combination of C and the Lua programming languages.The experiments are run on a cluster of 29 bi-quad-core Xeon machines, each with 8 GB of RAM and interconnected using a switched 1 Gbps network.

The workload consists of a trace of HTTP requests to the sites hosting the FIFA World Cup'98 website [1]. The data was collected for more than 80 days, including during the competition finale, and contains 240 million requests (events) over 32 different sites spread non-uniformely across the globe.

We start by analyzing the properties of the trace. Instead of fully characterizing the trace, which was done in detail in [1], here we focus just on the metrics pertinent to this paper. Next, we evaluate TOPiCo in terms of resource efficiency and closeness to the ideal $\mathcal{G}$ as computed by an omniscient entity.

5.1 Workload

Figure 1 shows the evolution of the number of sites and events during the competition. From the 80 days available in the trace, in the rest of the evaluation we focus just on days

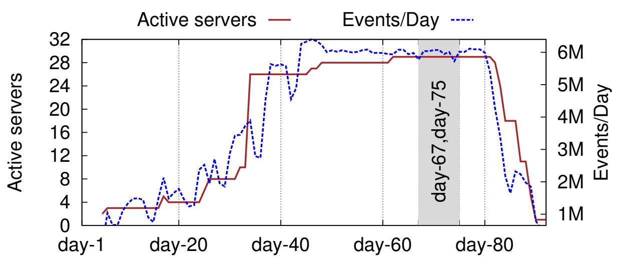

Figure 1: Active servers vs total events per day across active the servers. The gray area highlights the period considered in the rest of the evaluation.

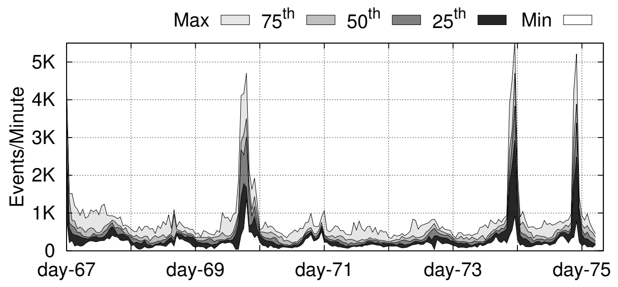

Figure 2: Events per minute across all active 29 servers between day-67 and day-75. There are more than 48 million events during this period.

67 to 75 which correspond to the competition finale. The reason for this is not only to run the experiments in a more reasonable timeframe, but also because this is where the highest load happens, and hence where TOPiCo becomes more interesting. During this peak period there are 29 active sites and around 6 million events per day.

In Figure 2, we show the overall number of events per minute across all sites for this peak period. We use a representation based on stacked percentiles throughout this section. The white bar at the bottom represents the minimum value, the pale grey on top the maximal value. Intermediate shades of grey represent the 25th, 50th -the median-, and 75th percentiles. For instance, the median number of events per minute around day-70 (during the peak) is 3,000. Clearly, there are peaks in load which will affect not only resource consumption but also the closeness of the computed $\mathcal{G}_i$ to the ideal $\mathcal{G}$.

The efficiency of TOPiCo, and in general of any top-k frequent items algorithm, depends on the distribution skew of events popularity. In fact, if all events where equally popular, computing the most popular ones would be not only impractical but also useless. Figure 3 depicts the distribution of event popularity for several days. as it is possible to observe, the distribution is mildly skewed meaning that the difference in popularity on the most popular items is only moderate. This implies that, sometimes, the list of candidates to enter the top-k might grow large thus affecting the efficiency of TOPiCo. Our experiments evaluating the efficiency of TOPiCo confirm this observation.

We complete the characterization of the workload by assessing how the composition of the top-k evolves over time.

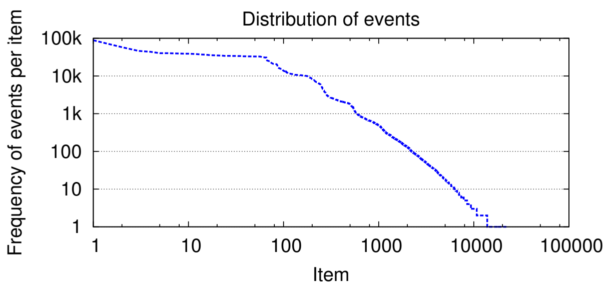

Figure 3: Distribution of event popularity across several days (the days not shown follow the same pattern). Note that the plot is log-log.

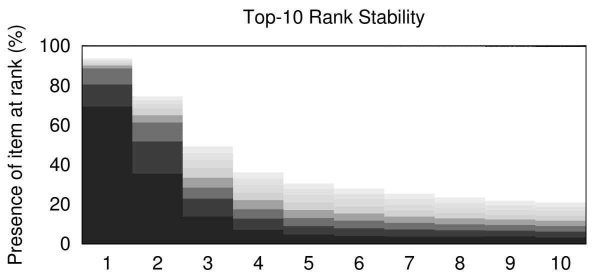

Figure 5: Rank stability showing the percentage of time a position in the rank is occupied by a given item.

For the sake of visualization, we only show results for K=10 and a sliding window of 20 seconds. A total of 311 different items appear in the top-10 during the time period between day-67 and day-75. Different perspectives of the same data are shown in Figure 4 and Figure 5. Each unique item is associated with a unique color. Figure 4 shows which items made it to the top-10 over time. Clearly some items are close to the topmost popular, while others stay mostly close to the bottom. This indicates that there is some stability on the most popular items. Figure 5 confirms this observation by showing the distribution of the time a given position in the top-k is occupied by a given item. For instance, we can see that a given item is the most popular (top-1) more than 60% of the time.

5.2 TOPiCo

We now focus on assessing TOPiCo when subject to the workload described above. Unless stated otherwise, presented results are the average over all sites with the following parameters: k=20, the size of the window is 15 seconds and the update period δ is 5 seconds. We start by observing how effective TOPiCo is in reducing the number of entries exchanged among sites. A naive approach would always broadcast the full $\mathcal{L}_i$ regardless of the workload. TOPiCo on the other hand exchanges just the minimal number of items necessary to compute correctly $\mathcal{G}_i$. These results are show in Figure 6. At Figure 6(top), we show the number of items sent as a fraction of the total number of items in $\mathcal{L}_i$. Clearly, TOPiCo is able to adapt the number of items sent accordingly to the workload. Still, sometimes up to 80%

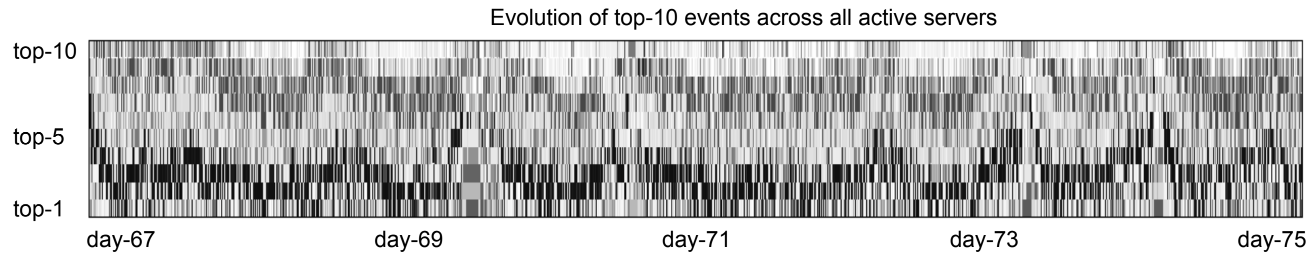

Figure 4: Trends of top-10 ranks computed by an omniscient observer over the 29 active servers. The sliding window size is 20 seconds. 311 unique events compete for a spot in the top-10 ranks. We notice clear trends and the extreme volatility of the spots at lower ranks.

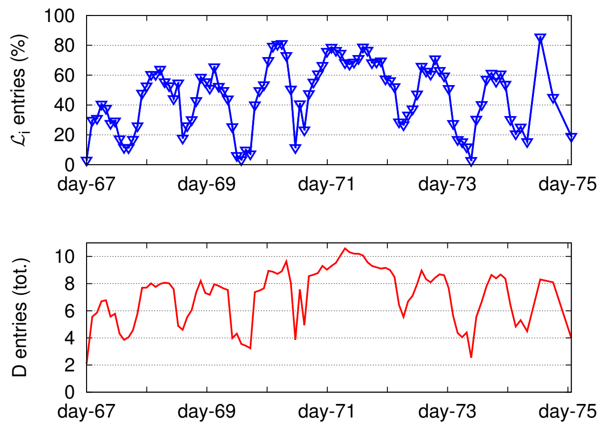

Figure 6: Cost of TOPiCo in terms of entries sent to other nodes as a function of the entries available in $\mathcal{L}_i$ (above) and as D entries (below).

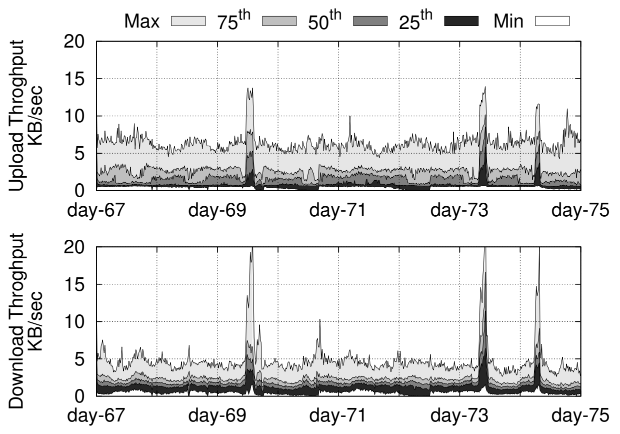

Figure 7: Throughput: upload (top) and download (bottom).

of the items in $\mathcal{L}_i$ need to be exchanged. Note that this is not a limitation of TOPiCo itself, but due to the intrinsic nature of the workload. In fact, as one can confirm in Figure 3, the difference in the frequency of the most popular items is small, implying that the number of candidates for $\mathcal{G}_i$ becomes potentially large. We expect TOPiCo to be able to significantly reduce the number of items sent when faced with more skewed workloads. Figure 6(bottom) complements this by showing, the number of additional items (D) that need to be sent. Even with this workload, the largest number of items exchanged is just 31 ($k + D = 20 + 11$) around day 71. Considering that for each item, we just send the item identifier and its frequency, the size of the exchanged list is still very small.

We confirm this by observing the download and upload throughput of sites, as shown in Figure 7. As expected, this is mostly affected by the arrival of events depicted in Figure 2. Both upload and download bandwidth usage remain under 10KB/sec most of the time which is quite small on modern infrastructures. The fact that few sites upload significantly more than the majority (notice the difference between the max and 75^{th} percentile on Figure 7) is because the reception of events is not uniformly spread across sites, causing some sites to maintain larger $\mathcal{L}_i$ and with more similar frequencies near the top, hence requiring to transmit more data.

We now focus on the closeness between $\mathcal{G}_i$, computed locally at each site, and $\mathcal{G}$ computed by an hypothetical omniscient observer. To this end we use the normalized Kendall-Tau rank distance which gives the pairwise differences between two ranked lists. A distance of zero means the lists are equal, while a distance of one means complete disagreement in the rankings. Figure 8 shows the normalized Kendall-Tau distance between $\mathcal{G}$ and $\mathcal{G}_i$. Here we split the servers according to their geographical location: Europe, US-EastCoast, US-Central and US-WestCoast. The reason for this is to observe how the geographical location, and hence, different access patterns affect the distance to $\mathcal{G}$. As shown, the distance to the omniscient $\mathcal{G}$ is roughly the same across all regions meaning TOPiCo achieves good results regardless of the composition of $\mathcal{L}_i$ s in a particular region.

Then, we investigate how the Kendal-Tau distance distance between $\mathcal{G}$ and $\mathcal{G}_i$ performs under peak hours. We choose the peak hours between day-69 and day-70 (Figure 2). In this time window, there are as many as 4,500 messages per minute. Figure 9 reports our result. We observe how TOPiCo performs similarly to the previous scenario, stating the benefits of our approach even under heavy load.

Finally, we show the delay between $\mathcal{G}_i$, computed locally at each site, and the real instantaneous $\mathcal{G}$. As before, we consider sites in different geographical regions. Results are presented at Figure 10 from day-67 to day-75. Again, the

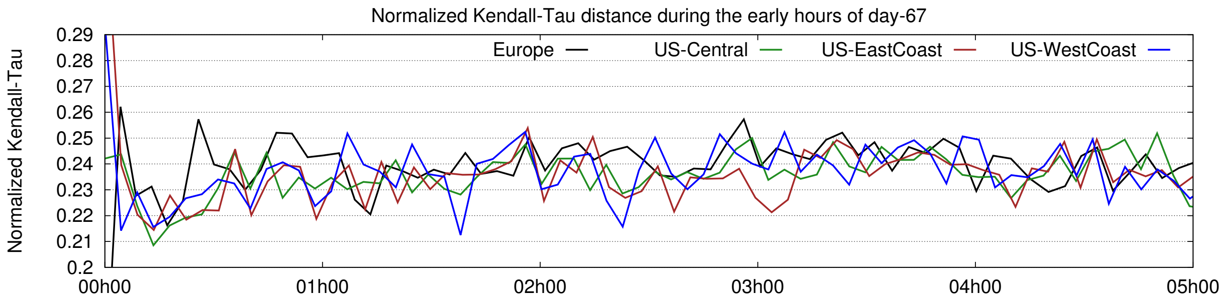

Figure 8: Kendall-Tau rank distance between $\mathcal{G}_i$ and $\mathcal{G}$ for 4 nodes in different regions. Results focus on the first 5 hours of day-67. A distance of 0 indicates identical rankings.

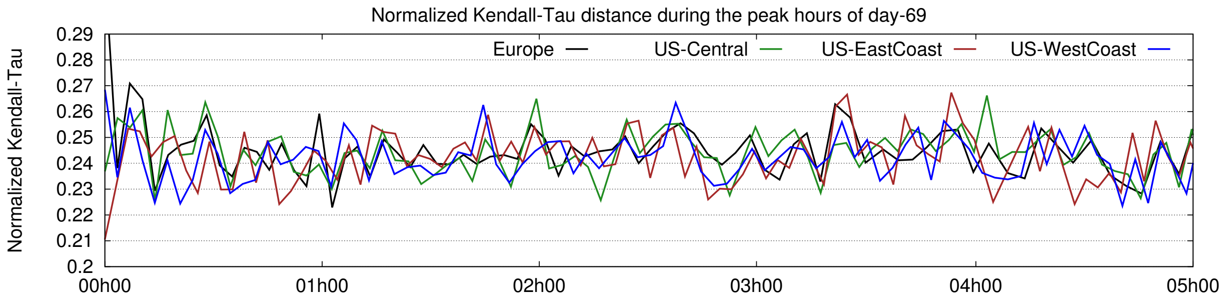

Figure 9: Kendall-Tau rank distance between $\mathcal{G}_i$ and $\mathcal{G}$ for 4 nodes in different regions. Results focus on the peak hours between day-69 and day-70.

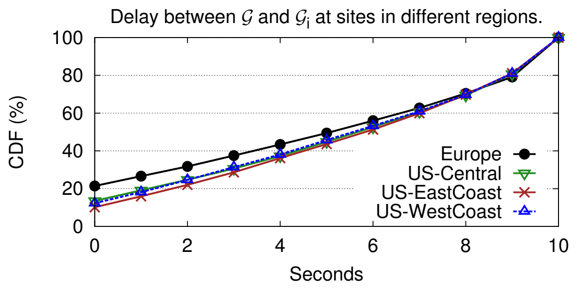

Figure 10: Distribution of delay between $\mathcal{G}_i$ and $\mathcal{G}$ for 4 nodes in different regions.

difference across regions is negligible meaning TOPiCo is able to dilute the local differences when computing $\mathcal{G}$. Half of the time, sites experience a latency smaller than the update period δ. Naturally, by decreasing δ we would observe a reduction in the latency at the expense of bandwidth.

Overall, the evaluation conducted allows us to conclude that TOPiCo delivers its promises: a lightweight, adaptable algorithm for computing the top-k most frequent items across a set of geographically distributed sites.

6. DISCUSSION AND CONCLUSION

In this paper we presented TOPiCo, a lightweight protocol for computing the top-k frequent items over several geographically dispersed event streams. TOPiCo works by selecting, locally at each site, the minimal amount of items that need to be exchanged such that each site is able to build a $\mathcal{G}_i$ close to the ideal $\mathcal{G}$ as observed by an omniscient observer. We provide a correctness proof of the algorithm and evaluate it in a real implementation with a real workload. The results confirm TOPiCo as a resource efficient approach able to adapt to variations in the workload.

In the present work we have not considered site failures. However, as TOPiCo relies only on local knowledge, and does not require any form of coordination among site, it is suited to work on an environment where sites may fail. Assessing TOPiCo behavior and any potential adjustments required to tolerate faults is part of our future plans.

Besides, we assume the existence of a broadcast primitive to disseminate information to all sites. The absence of an underlying multicast primitive results often in broadcast being implemented as a series of unicast calls, which limits the scalability of the system. We plan to overcome these limitations by extending TOPiCo to disseminate information using epidemic/gossip based protocols. Such protocols are known to be highly scalable but also robust to failures which aligns very well with the goals of TOPiCo we have in mind.

7. ACKNOWLEDGEMENTS

The authors are grateful to Flavio Junqueira for the initial discussions around this work. The research leading to this publication was partly funded by the European Commission's FP7 under grant agreement number 318809, LEADS project and 619606, LeanBigData, Ultra-Scalable and Ultra-Efficient Integrated and Visual Big Data Analytics project.

8. REFERENCES

[1] Arlitt, M., and Jin, T. A workload characterization study of the 1998 world cup web site. *Network, IEEE 14*, 3 (2000), 30–37.

[2] Babcock, B., and Olston, C. Distributed top-k monitoring. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data* (New York, NY, USA, 2003), SIGMOD '03, ACM, pp. 28–39.

[3] Brenna, L., Gehrke, J., Hong, M., and Johansen, D. Distributed event stream processing with non-deterministic finite automata. In *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems* (New York, NY, USA, 2009), DEBS '09, ACM, pp. 3:1–3:12.

[4] Cao, P., and Wang, Z. Efficient top-k query calculation in distributed networks. In *Proceedings of the Twenty-third Annual ACM Symposium on Principles of Distributed Computing* (New York, NY, USA, 2004), PODC '04, ACM, pp. 206–215.

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

[6] Culhane, W., Jayaram, K. R., and Eugster, P. Fast, expressive top-k matching. In *Proceedings of the 15th International Middleware Conference* (New York, NY, USA, 2014), Middleware '14, ACM, pp. 73–84.

[7] Demaine, E. D., López-Ortiz, A., and Munro, J. I. Frequency estimation of internet packet streams with limited space. In *Proceedings of the 10th Annual European Symposium on Algorithms* (London, UK, UK, 2002), ESA '02, Springer-Verlag, pp. 348–360.

[8] Fagin, R., Kumar, R., and Sivakumar, D. Comparing top k lists. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms* (Philadelphia, PA, USA, 2003), SODA '03, Society for Industrial and Applied Mathematics, pp. 28–36.

[9] Fagin, R., Lotem, A., and Naor, M. Optimal aggregation algorithms for middleware. In *Proceedings of the Twentieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems* (New York, NY, USA, 2001), PODS '01, ACM, pp. 102–113.

[10] Guerrieri, A., Montresor, A., and Velegrakis, Y. Top-k item identification on dynamic and distributed datasets. In *Euro-Par 2014 Parallel Processing*, F. Silva, I. Dutra, and V. Santos Costa, Eds., vol. 8632 of *Lecture Notes in Computer Science*. Springer International Publishing, 2014, pp. 270–281.

[11] Guntzer, J., Balke, W.-T., and Kiessling, W. Towards efficient multi-feature queries in heterogeneous environments. In *Proceedings of the International Conference on Information Technology: Coding and Computing* (Washington, DC, USA, 2001), ITCC '01, IEEE Computer Society, pp. 622–.

[12] Hirzel, M. Partition and compose: Parallel complex event processing. In *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems* (New York, NY, USA, 2012), DEBS '12, ACM, pp. 191–200.

[13] Ilyas, I. F., Beskales, G., and Soliman, M. A. A survey of top-k query processing techniques in relational database systems. *ACM Comput. Surv. 40*, 4 (Oct. 2008), 11:1–11:58.

[14] Lahiri, B., Chandrashekar, J., and Tirthapura, S. Space-efficient tracking of persistent items in a massive data stream. In *Proceedings of the 5th ACM International Conference on Distributed Event-based System* (New York, NY, USA, 2011), DEBS '11, ACM, pp. 255–266.

[15] Lahiri, B., and Tirthapura, S. Identifying frequent items in a network using gossip. *Journal of Parallel and Distributed Computing 70*, 12 (2010), 1241 – 1253.

[16] Manjhi, A., Shkapenyuk, V., Dhamdhere, K., and Olston, C. Finding (recently) frequent items in distributed data streams. In *Proceedings of the 21st International Conference on Data Engineering* (Washington, DC, USA, 2005), ICDE '05, IEEE Computer Society, pp. 767–778.

[17] Michel, S., Triantafillou, P., and Weikum, G. KLEE : A Framework for Distributed Top-k Query Algorithms. *VLDB '05 - Proceedings of the 31st VLDB conference* (2005), 637–648.

[18] Misra, J., and Gries, D. Finding repeated elements. *Sci. Comput. Program. 2*, 2 (1982), 143–152.

[19] Sacha, J., and Montresor, A. Identifying frequent items in distributed data sets. *Computing 95*, 4 (Apr. 2013), 289–307.

[20] Singh, S., Estan, C., Varghese, G., and Savage, S. Automated worm fingerprinting. In *Proceedings of the 6th Conference on Symposium on Operating Systems Design & Implementation - Volume 6* (Berkeley, CA, USA, 2004), OSDI'04, USENIX Association, pp. 4–4.

[21] Theobald, M., Weikum, G., and Schenkel, R. Top-k query evaluation with probabilistic guarantees. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30* (2004), VLDB '04, VLDB Endowment, pp. 648–659.

[22] Tudoran, R., Nano, O., Santos, I., Costan, A., Soncu, H., Bougé, L., and Antoniu, G. Jetstream: Enabling high performance event streaming across cloud data-centers. In *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems* (New York, NY, USA, 2014), DEBS '14, ACM, pp. 23–34.

[23] Vitter, J. S. Random sampling with a reservoir. *ACM Transactions on Mathematical Software 11*, 1 (1985).

[24] Wang, X., Candan, K. S., and Song, J. Complex pattern ranking (cpr): Evaluating top-k pattern queries over event streams. In *Proceedings of the 5th ACM International Conference on Distributed Event-based System* (New York, NY, USA, 2011), DEBS '11, ACM, pp. 279–290.

[25] Weigert, S., Hiltunen, M. A., and Fetzer, C. Community-based analysis of netflow for early detection of security incidents. In *Proceedings of the 25th International Conference on Large Installation System Administration* (Berkeley, CA, USA, 2011), LISA'11, USENIX Association.

[26] Wong, R. C.-W., and Fu, A. W.-C. Mining top-k frequent itemset from data streams. *Journal of Data Mining and Knowledge Discovery 13*, 2 (2006), 193–217.