Continuous Top-k Monitoring on Document Streams

Dr.K.Srinivas Babu1  G.Sumad
1Professor Department of Computer Science and Engineering  2M.Tech

Abstract—The efficient processing of document streams plays an important role in many information filtering systems. Emerging applications, such as news update filtering and social network notifications, demand presenting end-users with the most relevant content to their preferences. In this work, user preferences are indicated by a set of keywords. A central server monitors the document stream and continuously reports to each user the top-k documents that are most relevant to her keywords. Our objective is to support large numbers of users and high stream rates, while refreshing the top-k results almost instantaneously. Our solution abandons the traditional frequency-ordered indexing approach. Instead, it follows an identifier-ordering paradigm that suits better the nature of the problem. When complemented with a novel, locally adaptive technique, our method offers (i) proven optimality w.r.t. the number of considered queries per stream event, and (ii) an order of magnitude shorter response time (i.e., time to refresh the query results) than the current state-of-the-art.

Keywords—Top-k query, continuous query, document stream

I. INTRODUCTION

In the era of big data, the amount of information made available to users far exceeds their capacity to discover and understand it. For instance, a user on Twitter may receive an overwhelming volume of notifications if her message is retweeted by too many people in a short period. Moreover, the timeliness of information filtering and delivery is of great importance. For example, a user would like to receive instant updates of the hottest topics on social news and entertainment websites (e.g., on reddit.com). Thus, the efficient filtering and monitoring of rapid streams is key to many emerging applications.

We consider continuous top-k queries on documents (CTQDs), a topic which has received a lot of attention recently [1], [2], [3]. In this context, a central server monitors a document stream and hosts CTQDs from various users. Each CTQD specifies a set of keywords, as explicitly given by the issuing user or extracted from her online behavior [4], [5]. The task of the server is to continuously refresh for every CTQD the top-k most relevant documents to the keywords, as new documents stream in and old ones become too stale to be of interest.

The investment decisions of a stock broker are very sensitive to news about the stocks in her portfolio. To enable timely decisions, presenting her with the most relevant news as soon as they become available is key to the success of the notification system. Similar applications can be found in monitoring live Web content, such as RSS/news feeds, blog entries, posts on social media, etc. Widely available notification systems, such as Google Alerts (google.com/alerts) and Yahoo! Alerts (alerts.yahoo.com), attest to the significance of these applications. On the other hand, these systems either work in a semi-offline manner by delivering periodic updates (e.g., daily) or allow for coarse filtering only (e.g., based on general topics, rather than sets of specific keywords). Another application domain for CTQDs are microblog real-time search services [6], [7]. Currently, these services allow the user to query (in an on-demand, one-off way) for posts that match a set of keywords. CTQDs could extend the functionality of these services by offering continuous monitoring/notifications about new posts that match the keywords.

In traditional text search, there are snapshot (i.e., one-off) top-k queries over static document collections. The inverted file is the standard index to organize documents [8]. It comprises a list for every term in the dictionary; the list for a term holds an entry for each document that contains the term. By sorting the lists in decreasing term frequency, and with appropriate use of thresholding (e.g., [9]), a snapshot query can be answered by processing only the top parts of the relevant lists. Due to the said sorting, we refer to that paradigm as frequency-ordering. This common practice for snapshot queries has been followed by most approaches for continuous top-k search, albeit adapted to the “standing” nature of the continuous queries and the highly dynamic characteristics of the document stream, e.g., [1], [2].

In this work, we depart from frequency-ordering, and adopt a different paradigm, namely, identifier-ordering (ID-ordering). Past studies on
snapshot top-k queries revealed that, for sparse types of data, it may be more effective to sort the lists of the inverted file by document ID [10], thus enabling “jumps” within the relevant lists, i.e., disregarding contiguous fractions of the lists. This is an interesting fact, which however is not directly applicable to continuous top-k queries. An application of ID-ordering to document streams would incur costly index maintenance, and also it would require repetitive query reevaluation, as it entails no mechanism to reuse past query results in response to updates.

We propose an ID-ordering methodology for CTQDs. Our methodology involves three dimensions. First, we reverse the role of the documents and the queries. That is, we index the (relatively static) queries and probe the streaming documents against that index, in order to eliminate the need for index maintenance due to stream events. The general idea of indexing the queries instead of the data in a streaming context is commonly referred to as query indexing, and has been used for many types of continuous queries (e.g., [11]). Second, since we index user queries which, unlike the documents, typically comprise just a few terms (i.e., they are hugely sparse), we may effectively apply ID-ordering to the query index. The adaptation of ID-ordering to a query index, however, is far from trivial and requires a careful redesign of its inner workings, as we explain in Section 4.2. By incorporating the first two dimensions, we already have a preliminary CTQD method (albeit just a stepping stone to our complete, most comprehensive solution), termed Reverse ID-Ordering (RIO). RIO is already faster than existing CTQD approaches, but we do not stop there. Third, we complement RIO with a novel, locally adaptive technique that produces tighter processing bounds. This technique renders the overall CTQD method optimal w.r.t. the number of considered queries per stream event, i.e., we prove that it computes the score of an arriving document w.r.t. the smallest possible number of queries, for any algorithm that follows the ID-ordering paradigm and guarantees correctness. The resulting method is our most advanced technique, called Minimal RIO (MRIO).

Through an extensive experimental evaluation with streams of real documents, we demonstrate that MRIO outperforms the current state-of-the-art CTQD solution by an order of magnitude. Furthermore, the “internal” comparison between MRIO and RIO reveals that the vast performance improvements achieved are primarily due to the third dimension sketched above, i.e., due to our locally adaptive technique. The contributions we make in this paper are summarised as follows:

- Our advanced approach (MRIO) outperforms the current state-of-the-art by one order of magnitude.
- MRIO employs novel bounds that offer proven optimality w.r.t. the number of considered queries per stream event.
- MRIO is more than two times faster than RIO, demonstrating that a skillful adaptation of ID-ordering to CTQDs alone (as in RIO) is not enough to derive the improvements achieved in this work.
- We further improve the performance of MRIO by restructuring its query index (i.e., rearranging the queries inside) to better exploit locality and strengthen the pruning effectiveness of its bounds.
- Our evaluation has a broader experimental value too, because it involves (besides the state-of-the-art for CTQDs) methods for different formulations, which perform competitively, and were never put in the same testbed before.

II. RELATED WORK

In information filtering the objective is to remove from an information stream those items that are of no interest to the end users. Information filtering approaches have been studied for text streams [12], however, their focus is to determine an appropriate relevance threshold, based on the user’s profile and the stream’s characteristics [13]. The actual filtering involves fixed thresholds (and therefore binary relevance assessments per stream item), rather than relative similarity and ranking.

Publish—subscribe is a messaging pattern where the publishers of messages categorize their messages into classes, and the subscribers receive only those messages that fall in their classes of interest [14], [15]. Unlike CTQD, there is typically a set of predefined classes (instead of terms) and there is no notion of relative ranking. [16] does consider relative similarity, however, its goal is to identify the k most relevant queries for every newly published message. [17] proposes a probabilistic algorithm that keeps a select subset of the messages in a sliding window to support approximate top-k processing. Still in the publish—subscribe setting, [18] considers the social annotation of news articles. Specifically, given a set of news stories (documents), it maintains for each of them the k most related tweets posted. Although in the documents (news stories) play the role of the streaming queries, it could be applied to our setting (by treating user queries as news stories), although it is not tailored to it. We include this method in our experiments, abbreviated as TPS (for top-k publish—subscribe).
The top-k query is relevant to our work. Given a set of options and a scoring function defined over their attributes, the goal is to report the k options with the highest scores. Top-k processing methods have been extensively studied in relational databases; [19] offers an extensive survey. Among them, the threshold algorithm [9] is central to our competitors. It assumes that the options are indexed by a number of lists, each of which is responsible for one option attribute, and keeps options sorted in descending order of that attribute. The main idea is to consider options from the sorted lists in a round-robin fashion and maintain an upper bound (threshold) for the score of any unseen option. The algorithm terminates when the kth best option found so far scores no lower than the threshold.

In the context of text search engines, similarity search is typically framed as a top-k problem over a set of documents. Terms (in queries and documents) are treated as attributes, weighted based on a standard scheme (e.g., tf-idf or Okapi BM25). The score of a document for a query is defined as a function over their common terms, such as cosine similarity. To facilitate search, the documents are indexed by an inverted file; [8] surveys different types of inverted files and query processing techniques. The inverted file includes a sorted list per term. In the frequency-ordering paradigm, the sorting key is term frequency (weight), whereas in ID-ordering it is the document ID. In the former case, processing follows similar principles to the threshold algorithm in order to consider only the top parts of the sorted lists. In the latter case, the lists are read in their entirety but jumps over ID ranges are made possible; in Section 4.1, we describe in more detail the most efficient processing approach in this paradigm [10], [20].

Continuous versions of the top-k query have also been studied. Top-k monitoring was originally addressed over a stream of low-dimensional records [21], [22]. The proposed methods relied on spatial indices and geometric reasoning (e.g., dual space transformations), and were thus tailored to data in just a handful of dimensions. Bound by the dimensionality curse, these approaches are not applicable to document streams, because if terms were dealt with as attributes, dimensionality would be in the order of hundreds of thousands.

Rao et al. [23] consider streams of documents, but address a special version of continuous top-k queries where the query weights are equal (equivalently, the query terms are unweighted). In this version of the problem, if the search terms in a query q are a superset of those in another q0, then the score of a document w.r.t. q is always larger than its score w.r.t. q0. This means that if we compute the score of a stream document d w.r.t. q, and that score is already smaller than the score of the kth document in the result of q0, we can directly infer that q0 is not affected by d. The proposed solution utilizes this “coverage” relationship between queries to safely ignore some of them when a document streams in. It is inapplicable to our problem, where query weights are generally not equal. Even if an extension were possible, the chances of an ad-hoc user query being completely covered by another would be too slim.

Closest related to our work are methods for continuous top-k queries (with ad-hoc term weights) on document streams. [2] assumes the sliding window model and indexes the valid documents by a (frequency-ordered) inverted file. It uses the threshold algorithm to compute the initial top-k results, and maintains pointers in the sorted lists so as to resume processing from these positions when result refill is necessary. [1] proposes an approach that also relies on frequency-ordering and the threshold algorithm, but indexes the queries instead of the stream documents. It is shown to outperform [2] and is the current state-of-the-art.

We refer to it as reverse threshold algorithm (RTA). The same propose an approximate k-nearest neighbor monitoring technique for streams of low-dimensional points. However, their solution is inapplicable to CTQDs, because even if stream documents were mapped to points in term space, their dimensionality would be in the order of many thousands. Zhang et al. [26] study a shared processing framework for multiple aggregation queries on a stream. It is an interesting idea to share computations among queries. However, this work is inapplicable to CTQDs because it cannot handle weighted sum aggregates for arbitrary weights. That is, even if two CTQDs share some common terms, their respective weights for these terms are generally different.

III. PRELIMINARIES

In this section, we first define the similarity metric between queries and documents, and present the model of focusing on the fresher stream content. Next, we formalize the continuous top-k query on documents.

Similarity Measure

We treat the query (i.e., the set of keywords it specifies) and the documents as vectors. Letting T be the dictionary of all terms, a query or a document vector includes one weight per term in the dictionary.
That is, a query q and a document d are represented as

\[ q \equiv \{w_1; w_2; \ldots; w_{|T|}\}; \quad d \equiv \{f_1; f_2; \ldots; f_{|T|}\}; \]

The term weights wi and fi can be assigned to queries and documents, respectively, using any standard weighting scheme, such as tf-idf or Okapi BM25. Without loss of generality, we assume that all query and document vectors are normalized to unit magnitude. Note that user queries typically include just a few terms, which means that the query vectors are extremely sparse (i.e., include numerous zero values). The documents, too, include just a fraction of the terms in the dictionary, thus, there are many zeros in their vectors.

As per common practice [27], the textual similarity between a query q and a document d is defined as the cosine similarity of their vectors.

\[ c(q,d) = \frac{d^T q}{|d||q|} \]

the higher its value, the greater the textual similarity.

Document Freshness

In stream monitoring applications, the freshness of information is essential. Hence, a focusing on the most recent stream documents is required. The two prevalent formulations to achieve this focusing are the sliding window and the decay model. A sliding window only considers as valid the documents that arrived most recently; the sliding window has a fixed number of documents to consider, which means that the documents that arrived within a fixed number of time units before current time (time-based). On the other hand, in the decay model the score of the documents drops over time by applying a decay function, so that the more recent documents are favored in query answering. Our work applies to both models, but the latter (and in particular its forward-decay version) is better suited to our targeted applications, as suggested in [3], [28]. The forward-decay model works as follows.

Consider a document d with arrival time \( t_d \) and cosine similarity \( c_d q; d^P \) to query q. The score of d w.r.t. q is defined as

\[ S_d q; d^P = c_d q; d^P \exp(-Z d t_d) \]

where \( Z \) is an application-specified decay parameter, and \( d t_d \) is the difference between the document’s arrival time \( t_d \) and a reference time \( t_0 \) in the past (e.g., the system startup time). Unlike the vanilla decay model, this forward-decay formula associates an invariable score to each arriving document (that is, \( S_d q; d^P \) does not change over time) while at the same time it effectively captures the freshness requirement of streaming applications [28]. Note that in this model, arriving documents receive increasingly larger scores as time goes by.

Problem Definition

A stream of documents flows into a central processing server, which hosts a set of CTQDs. Each CTQD specifies a set of keywords (modeled as a query vector q) and a positive integer k. For the sake of notation, we denote by \( m \) the number of keywords it specifies. The result of a CTQD includes the k stream documents with the highest scores \( S_d q; d^P \) seen so far. The task of the stream server is to update all query results as new documents arrive. Document arrivals are referred to as stream events. The primary performance metric in our work is the time required to refresh (update) all CTQD results in response to stream events.

Although in our default setting document arrivals are the only type of stream events, in Section 6 we consider the handling of other events types, such as query insertions, query deletions, and document expirations. The handling of document expirations enables, among others, the application of our framework to the sliding window model too.

IV. ID-ORDERING TECHNIQUES

Index & Query Processing for Snapshot Queries

In this section, we overview the ID-ordering paradigm for snapshot queries. The documents are indexed by an inverted file, comprising a list \( L_i \) for every term \( t_i \) in the dictionary. \( L_i \) holds an entry \( h dID; f i i \) for every document that includes term \( t_i \) (where \( dID \) is the ID of the document, and \( f i \) its weight for term \( t_i \)). All lists are sorted in ascending document ID. The execution strategy to process a (snapshot) query q on this index evaluates the documents one after another from the sorted lists, but it performs “jumps” over zones of document IDs. The most efficient processing approach is Weighted AND [10], [20]; in the following we refer to this approach.

The maximum \( f i \) value in each list is pre-computed and stored with it – we denote it as \( m_i \). Posed a query, the algorithm executes in a number of iterations involving only the relevant lists. For every list \( L_i \), a cursor \( c_i \) is used to store the ID of the next unconsidered document in the list. Assume
that the query involves terms $t_1; t_2; \ldots; t_m$. At the beginning of an iteration, the processing order among the lists is decided based on their $c_i$, i.e., by placing first the list whose cursor points at the smallest document ID, then the list whose cursor points at the next smallest document ID, etc. Assume that the processing order is $L_1 ! L_2 ! \ldots ! L_m$ (equivalently, in the beginning of the iteration $c_1 \leq c_2 \leq \ldots \leq c_m$). The invariant of the method is that, for every $i \geq 1$, any list after the $i$th in the processing order includes no entry for document IDs in $[c_1; c_i]$. The invariant of the method is that, for every $i \geq 1$, any list after the $i$th in the processing order includes no entry for document IDs in $[c_1; c_i]$. The invariant of the method is that, for every $i \geq 1$, any list after the $i$th in the processing order includes no entry for document IDs in $[c_1; c_i]$.

### V. CONCLUSION

In this paper, we propose a scalable framework for the processing of continuous top-$k$ queries on document streams. A CTQD continuously reports the $k$ most relevant documents to a set of keywords. CTQDs find application in many emerging applications, such as email and news filtering. Our preliminary approach, RIO, adapts the ID-ordering paradigm to the CTQD setting. An analysis on RIO reveals that the key factor that determines its performance is the number of iterations it executes. This motivates our advanced approach, MRIO, which not only reduces the number of iterations, but is proven to minimize it. We achieve this by introducing novel, locally adaptive bounds. Extensive experiments with streams of real documents demonstrate that MRIO is an order of magnitude faster than the previous state-of-the-art. A promising direction for future work is to extend our methodology to approximate top-$k$ queries.

### References


