Continuous Top-k Monitoring on Document Streams (Extended Abstract)

Leong Hou U [#1], Junjie Zhang [#2], Kyriakos Mouratidis [*3], and Ye Li [#4]

[#] *Department of Computer and Information Science, University of Macau*
Macau SAR
[1] ryanlhu@umac.mo, [2] zhjjie89@gmail.com, [4] yb47438@umac.mo

[*] *School of Information Systems, Singapore Management University*
Singapore
[3] kyriakos@smu.edu.sg

***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. The 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, and follows an identifier-ordering paradigm that suits better the nature of the problem. When complemented with a locally adaptive technique, our method offers (i) optimality w.r.t. the number of considered queries per stream event, and (ii) an order of magnitude shorter response time than the state-of-the-art.**

I. Introduction

In this work, we consider *continuous top-k queries on documents* (CTQDs), a topic which has received a lot of attention [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 behaviour. The task of the server is to continuously refresh for every CTQD the top-k most relevant documents, as new documents stream in and old ones become too stale.

To answer top-k queries over static document collections, the standard index used is the inverted file. In the ID-ordering paradigm, the term lists of the inverted file are sorted by document ID, thus enabling efficient "jumps" within the lists during query processing. This approach is not directly applicable to CTQDs. An application of ID-ordering to document streams would incur costly index maintenance, and also it would require repetitive query reevaluation.

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. **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. By incorporating the first two dimensions, we already have a preliminary CTQD method, termed *Reverse ID-Ordering* (RIO). **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 fewest possible queries, for any exact algorithm that follows the ID-ordering paradigm. The resulting method, called *Minimal RIO* (MRIO), outperforms the state-of-the-art by one order of magnitude.

II. 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 $\mathbf{q}$) and a positive integer k. The result of a CTQD includes the k stream documents with the highest scores $S(q,d)$ seen so far, with score defined as:

$$S(q,d) = c(q,d)/e^{-\lambda \Delta \tau_d} \quad (1)$$

where $c(q,d)$ is the cosine similarity of $\mathbf{q}$ and $\mathbf{d}$, τ_d is the arrival time of document $\mathbf{d}$ and λ is a decay parameter.

The task of the processing 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.

III. Solutions

All registered queries $\mathbf{Q}$ (in the processing server) are indexed by an inverted file, comprising a list L_i for every term t_i in the dictionary. L_i holds an entry $\langle q_{ID}, w_i \rangle$ for

every query that includes term t_i (where q_{ID} is the ID of the query, and w_i its preference weight for term t_i).

RIO. When a new document **d** arrives at the processing server, we check whether **d** can be included into the result of any registered query. For each arriving document, RIO 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 query in the list. Assume that the document involves terms $t_1, t_2, ..., t_m$. 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 query ID, then the list whose cursor points at the next smallest query ID, etc. Assume that the processing order is $L_1 \rightarrow L_2 \rightarrow ... \rightarrow L_m$ (equivalently, in the beginning of the iteration $c_1 \leq c_2 \leq ... \leq c_m$). The invariant of the method is that, for every $i \in [1, m)$, any list after the i-th in the processing order includes no entry for query IDs in $[c_1, c_i)$. That is, the score of every query ID in the zone $[c_1, c_{i+1})$ is upper bounded by:

$$UB(i) = \sum_{1 \leq j \leq i} f_j \cdot \max_{q \in Q} \{w_j / S_k(q)\} / e^{-\lambda \Delta \tau_d} \qquad (2)$$

where f_j is the weight for term t_j of document **d**. The upper bound requires the maximum normalized preference of all registered queries for term t_j. $S_k(q)$ is the normalized factor that denotes the score of the k-th best document of query **q**.

The algorithm identifies the smallest $i \in [1, m]$ for which $UB(i) \geq 1$ and sets the corresponding cursor c_i as the *pivot*. Effectively, all queries in the zone $[c_1, c_i)$ can be safely pruned. Thus, all cursors advance ("jump") to the first ID in their list that is no smaller than c_i. The process terminates when all cursors reach the end of their lists.

MRIO. The response time of RIO depends on the number of iterations (cf. Sec. 5.1 in [4]). An ideal way to improve performance is to reduce the number of iterations required. To achieve that, we need to perform as large jumps in the relevant lists as possible. In turn, what determines the length of the jumps is the tightness of the upper bounds $UB(i)$.

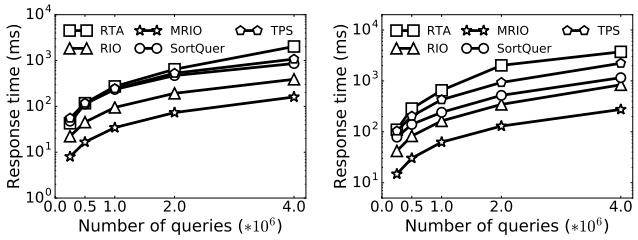
Assume that the processing order is $L_1 \rightarrow L_2 \rightarrow ... \rightarrow L_m$. Recall from the discussion on Equation 2 for standard ID-ordering that the i-th upper bound regards IDs in $[c_1, c_{i+1})$ (except for $i = m$ where the ID range is $[c_1, c_m]$) – it is exactly these IDs that are candidates for pruning when the i-th upper bound is considered. Thereby, the local, and thus tighter, upper bounds $UB^*(i)$ in MRIO are defined as:

$$UB^*(i) = \sum_{1 \leq j \leq i} f_j \max_{q \in i^{th} zone} \{w_j / S_k(q)\} / e^{-\lambda \Delta \tau_d} \qquad (3)$$

By replacing the upper bound condition $UB(i)$ by $UB^*(i)$, MRIO is proven to invoke the minimum possible number of iterations subject to the ID-ordering execution paradigm (cf. Lemma 2 in [4]). We consider three alternative implementations for $UB^*(i)$ (cf. Sec. 5.2 in [4]).

IV. Experiments

We evaluate RIO and MRIO against RTA [1], SortQuer [3], and TPS [5]. We use 7,012,610 actual Wikipedia pages to simulate the document stream. We experiment with two synthetic query workloads, *Connected* and *Uniform*, exhibiting different word co-occurrence frequencies.



(a) Wiki - Uniform (b) Wiki - Connected

Figure 1. Effect of number of queries

The response time of all methods increases, since the number of queries that are affected by a document arrival grows proportionally to their total number. The running time of MRIO is up to 8, 10, and 25 times shorter than TPS, SortQuer, and RTA, respectively.

V. Conclusion

We propose a scalable processing framework for continuous top-k queries on document streams. We index the queries (instead of the documents) and, in contrast to previous approaches, we adopt the identifier-ordering paradigm, enhanced with novel bounds. Our best algorithm is an order of magnitude faster than the previous state-of-the-art.

Acknowledgments

Leong Hou U was supported by MYRG-2016-00182-FST from UMAC RC and 61502548 from NSFC. Kyriakos Mouratidis was supported by the Singapore Ministry of Education (MOE) Academic Research Fund Tier 1 grant.

References

[1] P. Haghani, S. Michel, and K. Aberer, "The gist of everything new: personalized top-k processing over web 2.0 streams." in *CIKM*, 2010, pp. 489–498.

[2] K. Mouratidis and H. Pang, "Efficient evaluation of continuous text search queries," *IEEE Trans. Knowl. Data Eng.*, vol. 23, no. 10, pp. 1469–1482, 2011.

[3] N. Vouzoukidou, B. Amann, and V. Christophides, "Processing continuous text queries featuring non-homogeneous scoring functions." in *CIKM*, 2012, pp. 1065–1074.

[4] L. H. U, J. Zhang, K. Mouratidis, and Y. Li, "Continuous top-k monitoring on document streams," *IEEE Trans. Knowl. Data Eng.*, vol. 29, no. 5, pp. 991–1003, 2017.

[5] A. Shraer, M. Gurevich, M. Fontoura, and V. Josifovski, "Top-k publish-subscribe for social annotation of news," *PVLDB*, vol. 6, no. 6, pp. 385–396, 2013.