The Gist of Everything New: Personalized Top-k Processing over Web 2.0 Streams[*]

Parisa Haghani
EPFL
Lausanne, Switzerland
parisa.haghani@epfl.ch

Sebastian Michel
Saarland University
Saarbrücken, Germany
smichel@mmci.uni-saarland.de

Karl Aberer
EPFL
Lausanne, Switzerland
karl.aberer@epfl.ch

ABSTRACT

Web 2.0 portals have made content generation easier than ever with millions of users contributing news stories in form of posts in weblogs or short textual snippets as in Twitter. Efficient and effective filtering solutions are key to allow users stay tuned to this ever-growing ocean of information, releasing only relevant trickles of personal interest. In classical information filtering systems, user interests are formulated using standard IR techniques and data from all available information sources is filtered based on a predefined *absolute* quality-based threshold. In contrast to this restrictive approach which may still overwhelm the user with the returned stream of data, we envision a system which continuously keeps the user updated with only the top-k relevant new information. Freshness of data is guaranteed by considering it valid for a particular time interval, controlled by a sliding window. Considering relevance as *relative* to the existing pool of *new* information creates a highly dynamic setting. We present *POL-filter* which together with our maintenance module constitute an efficient solution to this kind of problem. We show by comprehensive performance evaluations using real world data, obtained from a weblog crawl, that our approach brings performance gains compared to state-of-the-art.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Search process*; H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing—*Indexing methods*; H.4.m [**Information Systems**]: Miscellaneous

[*]This work is partially supported by NCCR-MICS (grant number 5005-67322), the FP7 EU Project OKKAM (contract no.ICT-215032), and the German Research Foundation (DFG) Cluster of Excellence "Multimodal Computing and Interaction" (MMCI).

General Terms

Algorithms, Experimentation

Keywords

Top-k Query, Information Filtering, Data Streams

1. INTRODUCTION

The world has turned into one large-scale interconnected information system with millions of users. With the advent of Web 2.0, yesterday's end users are now content generators themselves and actively contribute to the Web. Each user action, for example uploading a picture, tagging a video, or commenting on a blog, can be interpreted as an event in a corresponding stream. Given the immense volume of this data and its vast diversity, there is a vital need for effective filtering methods which allow users to efficiently follow personally interesting information and stay tuned.

Currently popular methods place the filter on the data sources: mechanisms such as RSS and atom are used to notify users of newly published data on their favored weblogs or news portals. However, with the currently available functionalities, users can only decide to be notified of new posts on certain blogs or follow certain other users as in Twitter. This limits the number of subscriptions users make, as otherwise the amount of received information will be overwhelming for human processing. On the other hand, traditional information filtering systems [5, 27], aggregate all available information sources and allow users to specify their interests as profiles. Given a similarity measure between the data and the profiles, only data which passes a certain quality-based threshold is returned to the user. Although this diversifies the returned results as opposed to the previous method, it can easily result in flooding the user with returning too many data. Choosing a suitable threshold to avoid overwhelming the user or returning very few results is very hard due to the ever changing nature of incoming data. This calls for a system which deems relevance as *relative* to the existing pool of information [21], as opposed to *absolute* relevance. Furthermore, to account for the desire of consuming new information and to prohibit repeatedly returning highly relevant, but old information, data is considered valid for only a certain time interval, controlled by a sliding window. Note that in the context of Web 2.0, all information come with explicit temporal annotations e.g., *written at*, *uploaded at*, which makes them natural items of a temporal stream; therefore the definition of a sliding window is meaningful. The dynamism introduced as a result of considering relevance relative in a frequently changing information pool, as well as the scale of our envisioned system in handling huge number of users, poses fundamental new challenges which were nonexisting previously.

As an illustration, emphasizing the importance of the addressed problem, consider a small scale case where $100,000$ profiles are maintained at a single server. The naive approach consists of evaluating every profile against the incoming documents and re-evaluating the profiles upon result expiration from scratch. According to our experiments, these operations, disregarding the cost for indexing the documents, i.e., removing stopwords, calculating TF/IDF values take on average, orders of tens of milliseconds per document on a quad-core Intel Xeon CPU E5530 @2.4GHz machine. This means that the maximum supported rate of incoming documents would be in order of hundred documents per second which is relatively small, given that today only in Twitter, around 600 messages are produced per second [1].

1.1 Problem Statement

We consider a stream $\mathcal{S}$ of documents where each document is uniquely identified and consists of a weight vector, as in classical Vector Space Model, as well as its arrival time: $\mathbf{d} = \langle id, time, d\rangle$. Assuming m distinct terms available for content identification, d is an m-dimensional vector $d = (w_1, ..., w_m)$, where w_i is the weight assigned to the i-th term. Terms which do not appear in the document have a zero weight. Any of the usual scoring schemes such as the TF/IDF methodology can be used for assigning the weights. We further assume *in-order* streams; items arrive in the same order that they are generated. In most streaming scenarios, as well as ours, recent items are of more interest than old ones. This is captured by the sliding window model. A sliding window (W) is assumed over the stream and items are considered *valid* while they belong to this window. Sliding windows can be either count or time based, i.e., bounding the number of items either by count or focusing only on those that occurred in a particular time interval. Our solution can be applied to both types.

Similar to web search, we assume user interests, called *profiles*, are expressed as sets of terms with corresponding weights: We denote a profile by $p = (u_1, ..., u_m)$, where u_i specifies the importance of the i-th term to the user. The relevance of a document to a profile is determined by a scoring function: $sim(d, p) = g(f_{w_1}(u_1), ...f_{w_m}(u_m))$. We make the following assumptions regarding g and f:

- **Monotonicity**: We assume g and f_i's are monotone. $g(x_1, ..., x_m)$ is monotone if $g(x_1, ..., x_m) \leq g(x'_1, ..., x'_m)$ whenever $x_i \leq x'_i$ for all i and similarly for f_is.
- **Homogeneity**: We assume g and f_i's are homogeneous, i.e., they preserve the scalar multiplication operation: $g(ax_1, ..., ax_m) = a^r g(x_1, ..., x_m)$. For simplicity we assume that the homogeneity degree is 1: $r = 1$. However, our solution can be applied for higher degrees of homogeneity as well.

Note that taking the f functions as multiplication and g as summation, we arrive at the widely used cosine measure as the scoring function for normalized vectors.

We consider a main memory model and treat each profile as a top-k query. All queries should be continuously monitored to keep the users up-to-date while the valid pool of information changes due to arrival of new documents or expiration of old ones. This goal involves two main tasks: first is efficient and scalable profile filtering in order to avoid comparing an incoming document against the large set of *all* existing profiles. Second is maintaining the top-k results of each profile as the window slides and some documents become invalid. In both tasks we focus on efficiency which is a necessary step towards ensuring scalability.

[1]http://blog.twitter.com/2010/02/measuring-tweets.html

1.2 Contribution and Outline

In this paper we make the following contributions.

1. We design an efficient profile filtering algorithm, $POL-filter$, which refrains from processing all profiles. We show that our method is exact and all profiles which a new incoming document has a chance of being their top-k result are identified.
2. We use a skyline based method for result maintenance, but in order to avoid inserting *all* incoming documents to the result set, which is the case for in-order streams, we restrict the insertion criteria such that the size of the maintained result set remains small. We derive the necessary conditions to insure exact results.

This paper is organized as follows: Section 2 presents the related work. Section 3 briefly describes the general structure that we consider in this paper together with a baseline algorithm. Section 4 describes an efficient algorithm for profile filtering. Section 5 discusses the skyline-based algorithm for result maintenance which aims at avoiding re-evaluations to high extent. Section 6 presents the experimental evaluation and Section 7 concludes the paper.

2. RELATED WORK

Our envisioned problem is the conjunction of two fundamental areas of research: *Stream Processing* and *Information Filtering*. Information filtering or *selective dissemination of information* is considered dual to classical information retrieval. In information retrieval, similar to our scenario, users subscribe to a system with their favorite profiles and receive notification whenever a relevant item arrives at the system. Traditionally relevance is defined by a fixed threshold or a boolean model is used. Setting this threshold to a *suitable* value is a very difficult task: a low value causes the user to be overwhelmed with the amount of results returned, while a too high value results in too few or no results at all. Instead we aim at returning the top-k results with regard to a constantly changing pool of data. Information filtering has been considered in vector space [5, 27], boolean [28] and more recently *AWP* [25] models. As mentioned in Section 1.1, we also consider the vector space model due to its popularity in use and simplicity. In [27], a profile filtering scheme is proposed which is based on distinguishing the *significant* and *insignificant* terms of a profile based on the given threshold. Only the significant terms of a profile are indexed and they are selected in a way that completeness of results, with regard to the given threshold, is guaranteed. Note that this method is not applicable in our setting as we do not consider a fixed threshold for returning the results. In [5] the previous method, which reduces the cost of disk I/O at the expense of larger indices, is combined with a document batching process which takes advantage of the sparsity of the profile and document matrices and writes the partial similarity matrix to disk, improving the efficiency. Callan describes a document filtering system [8] based on the inference network model of information retrieval. Information filtering is essentially similar to bichromatic reverse nearest neighbor search (RNN). Given a database of points P, a set of query points Q, and a similarity measure between the members of P and Q, in bichromatic RNN search, with a query $q \in Q$ the goal is to find $p \in P$ which is closer to q than any other point of Q. Most proposed methods for this problem consider two dimensions. In [16] two separate R-trees are used as the index structure for RNN search. [24] considers the monochromatic version of RNN in two dimensions and is based on the geometric observation that the maximum number of RNN's in two dimensions for a query point is 6. Singh et al. in [23] propose an approximate method for RNN search in high dimensional data which first finds the NN's of a query point with the hope that its RNN is actually among them. In our setting we focus on exact results, as each query represents a user who wishes to receive complete answers.

Stream processing has been a hot topic in the past few years due to its suitability in modeling large number of today's applications which are not captured well with the models offered by traditional database systems (for comprehensible surveys see [4, 22]). A related problem in stream processing is top-k query answering over sliding windows. Mouratidis et al. [20] maintain a skyline [7] which represents the possible top-k candidates. Their solution is optimized for *fixed* queries and they focus on changes introduced by items timing out or new items arriving in. We also use the concept of skylines to maintain our result sets and prove the completeness of results returned given the fact that we do not maintain the *complete* skyline. Furthermore, their grid-based indexing structure is not usable in our setting, as we are considering high dimensional textual data which will cause an explosion in the index size in a partitioning structure. In a more general setting, [9] proposes indexing methods for answering *adhoc* top-k queries based on arrangements. Again, the dimensionality of the data in their setting is low, so their solution is not applicable to our problem. Jin et al. [14] consider top-k queries on uncertain streams where the data items are associated with existential probabilities. The authors in [11] focus on top-k monitoring over multiple non-synchronized streams, where complete score calculations are not possible.

The closest work in the literature to ours is the recent work by Mouratidis et al. [21] which considers processing continuous text search queries. They maintain the valid documents in inverted lists and execute a threshold algorithm. While this is similar to our setup, the key idea in their approach is to keep the state of the TA algorithm, for each query (i.e., profile), in a per-term index organized as a tree. Upon arrival of new documents, the tree is scanned for all potentially affected profiles and in case of a change in the score of the document at rank k, the thresholds are updated upwards. In the case of the removal of old documents the index lists' scan lines will be adapted downwards. The rationale behind this continuous adaptation of the scan lines is that tight bounds cause fewer documents having to be evaluated against the registered queries. However, these scan lines are too often not a good (tight) description of the actually more interesting score at rank k, leading to the problem that many profiles have to be checked for modification with almost every incoming document. An additional effect of the the scan line based indexing is the large number of potential candidates held in the resultset. We address both problems (profile indexing and result maintenance) in this work and have implemented the approach by Mouratidis et al. [21] and include it in our experimental evaluation.

Continuous k nearest neighbors (kNN) queries on data streams is also a related topic which has been considered in [17, 6]. Koudas et al. present Disc [17] for indexing high dimensional points using space filling curves to give approximate answers to kNN queries. On the contrary we aim at providing exact results. Böhm et al. in [6] consider a fixed number of queries and index queries instead of incoming tuples in a structure similar to VA files [26] to continuously provide exact answers in a sliding window model. They also maintain a skyline to decide which tuples should be kept, therefore minimizing the needed storage. Essentially all the above are similar to our problem, if we consider each profile as a continuous query. However none of the indexing methods which provide exact results are applicable to our setting due to the high dimensionality of textual data we are considering.

Mainly motivated by the wealth of news feeds and other online information streams, another problem is *Topic Detection and Tracking* (TDT) which has been extensively studied in the past few years [3, 2, 12]. The goal is to detect new events appearing in the data stream and tracking those events to later identify data which further discuss the same event. Mining common topics in multiple asynchronous text streams is considered in [1]. In another line of research related to Web 2.0 applications with temporal considerations, Hotho et al. [13] consider discovering topic-specific trends in folksonomies which are collections of resources tagged by users (such as Flickr or del.icio.us [2]). Their analysis is performed offline assuming the whole corpus of data is available and is based on the PageRank algorithm. Weblog evolution is considered in [18], where *time graphs* are introduced and used for community tracking again in an offline mode. In [19], the goal is to identify weblogs defined as *starters* and *followers* specified by certain linking relations in an efficient way. For a survey of temporal data analysis methods see [15] and the references within.

3. SYSTEM MODEL AND STRUCTURE

In this section we briefly describe the general structure that we consider. As mentioned in Section 1.1 we consider one data stream as the input to our system. We aim at providing real-time continuous exact results to the users, therefore results returned as the top-k most relevant documents for each profile should consist of the top-k results over *all* valid documents in the system at each instance of time. With small number of queries (profiles) or infrequent updates in the result set of each profile, all queries could be re-evaluated at certain times to this end. However, given large number of profiles, this solution does not scale to provide real-time monitoring of all profiles' results. Instead, we index the profiles and assess the suitability of a new document for each profile as the document arrives in the system. We avoid evaluating all profiles against a new incoming documents, by a *profile filtering component.* The profile filtering component receives a newly arrived document as input and returns a set of profiles which should be updated with regard to this document. As the baseline, we maintain an inverted list structure in the profile filtering: For each term t we keep a list of profiles which contain this term, i.e., weight of t is larger than zero for those profiles. We also maintain a hashtable of profiles, where profiles with a pointer to their current result set are stored. When a new document d arrives, we evaluate all entries of all inverted lists corresponding to a non-zero weighted term in d. Evaluating an entry corresponding to a profile p consists of calculating

[2]http://flickr.com and http://del.icio.us

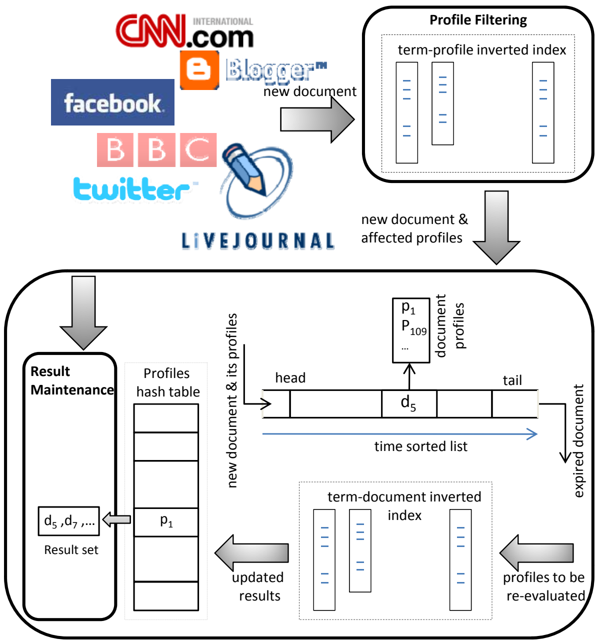

Figure 1: The Overall Structure

$sim(d, p)$ and comparing this with the current rank k result of p which can be known by accessing the profile hashtable. If the new document has a better similarity score, p's result set is updated with it.

As the window slides, some documents expire and the set of valid documents changes. Some profiles may need to be re-evaluated as the expired documents were part of their result set. The *result maintenance component* is concerned with efficiently performing this task. We keep a simple time-sorted list for tracking valid documents: newly arrived documents are inserted at the head and those which expire drop out from the tail. For each document we maintain the set of profiles to which this document is a top-k result. Therefore, when a document expires, the set of profiles which should be re-evaluated is known. The actual method for performing the re-evaluation is not a main concern of this paper. However, we assume sorted inverted lists are maintained such that the usual threshold algorithm (TA) [10] is employed for query evaluation. Figure 1 presents the general components and structures we consider.

4. EFFICIENT PROFILE FILTERING

Similar to the baseline described in Section 3, we use an inverted index of profiles to avoid examining all the existing profiles against the newly arriving documents. In contrast to the former approach, we utilize sorted versions of these lists. As we will describe in this Section, processing these sorted lists will enable early stopping to further reduce the number of examined profiles. Our method is similar to the well-known TA (threshold algorithm) [10] which is widely used in information retrieval.

As previously mentioned, we consider each profile as a continuous top-k query over the stream of incoming documents. Let $p.s$ denote the similarity score of the ranked k document with regard to profile p. Let $T = \{t_1, t_2, ..., t_m\}$ be the set of distinct terms considered for content identification of both profiles and documents. $p.u_i$ represents the corresponding weight assigned to term t_i in p. For each term t_i, we build a list l_i containing $\langle p.id, p.v_i \rangle$ pairs where $p.v_i = p.u_i/p.s$ and the list is sorted in decreasing order based on v_i values. $p.id$ denotes the unique identifier of profile p. A profile with l terms will only appear in l of such posting lists.

Assume a document d with the feature vector $d = (w_1, ...w_m)$ arrives in the system. We do sorted accesses in a round robin fashion to all posting lists l_i where $w_i > 0$. When a profile p is seen under one of these lists, we access the profile hash table for its complete weight vector and consequently calculate the similarity score between d and p. The result set of profile p is updated if

$$sim(d, p) > p.s$$

where as mentioned before $sim(d, p)$ satisfies the two conditions described in Section 1.1.

While accessing the sorted lists in a round robin fashion, we also check the following stopping condition. For a list l_i let $\underline{v_i}$ be the last observed value under sorted access. We stop the above procedure when $g(f_{w_1}(\underline{v_1}), ...f_{w_m}(\underline{v_m})) < 1$.

We call the above procedure *COL-filter* (Completely Ordered Lists) and show that it is complete: all profiles for which a new incoming document serves as a top-k result are identified before the stopping condition.

Theorem 1. COL-filter *identifies all profiles for which a new incoming document d is a top-k result.*

As g is monotone and the lists are sorted, reaching the stopping condition means that for any non processed profile p, $sim(p, d) < p.s$. For a detailed proof please see the appendix.

Since the number of lists which are processed could be much larger than the number of terms a profile has, we take into account the maximum number of terms a profile can have in calculating the stopping condition. Assume the maximum number of terms per profile is m'. The stopping condition could be checked per list, i.e., the procedure can stop processing one list while the other lists should still undergo the procedure. Let I be a subset of size m' of the lists under process. We define the following:

$$f^I_{w_i}(\underline{v_i}) = \begin{cases} f_{w_i}(\underline{v_i}) & \text{if } l_i \in I \\ 0 & \text{otherwise} \end{cases}$$

Also, let $\mathbb{I}_j$ be the set of all subsets I, where I includes l_j. The algorithm stops processing l_j if

$$max_{I \in \mathbb{I}_j} g(f^I_{w_1}(\underline{v_1}), ...f^I_{w_m}(\underline{v_m})) < 1$$

If g is symmetric, i.e., its value at any m-tuple of arguments is the same as its value at any permutation of that m-tuple, it is enough to do the test for one set I_{max} which contains the $(m' - 1)$ lists with the largest $f_{w_i}(\underline{v_i})$ values along with l_j. The general steps are shown in Algorithm 1.

While COL-filter enables early stopping and avoids accessing and assessing all profiles which appear in an inverted list of a term in an incoming document, it incurs high costs for maintaining the inverted lists. Contrary to standard information retrieval inverted lists, where the lists are static and change rarely, the sorted lists in COL-filter change frequently. This is because the values we use for sorting depend on the similarity scores of profiles which change with time, as new documents arrive or old ones expire and are removed from the system. Note that each time a profile p is updated, i.e., a new incoming document qualifies as its top-k result,

Algorithm 1: The COL filtering algorithm

```
ProfileFilter **Input**: $d = (w_1, ..., w_m)$
$toProcess = \emptyset$;
$toUpdate = \emptyset$;
if $w_i > 0$ then
    $toProcess$.add($l_i$);
while $toProcess \neq \emptyset$ do
    foreach list $l_i \in toProcess$ do
        $p = l_i$.getNext().getProfile();
        $v_i = p.v_i$;
        $p.s = p$.getScore($k$);
        if $sim(p, d) > p.s$ then
            $toUpdate$.add($p$);

    foreach list $l_j \in toProcess$ do
        if $max_{I \in \mathbb{I}_j} g(f^I_{w_1}(\underline{v_1}), ...f^I_{w_m}(\underline{v_m})) < 1$ then
            $toProcess$.remove($l_j$);
        if !$l_j$.hasNext() then
            $toProcess$.remove($l_j$);

return $toUpdate$;
```

$p.s$ changes. As a result $p.v_i = p.u_i/p.s$ changes, therefore p's corresponding tuples in all lists which p appeared in should be updated. As a consequence of the high dynamism inherent in the system, which is due to the high rate of incoming documents and their expiration, the number of necessary updates can be very high. The cost of maintaining the lists sorted can therefore overshadow the benefits of early stops. In the following, we propose a relaxation to completely sorting the lists, which requires significantly fewer number of updates and is cheaper to maintain.

4.1 Partially Ordered Lists

We aim at decreasing the cost of maintaining the sorted lists by grouping entries and ordering the entries only based on a fixed number of predefined boundaries instead of maintaining full order. These boundaries are then used to test the stopping condition. Our *Partially Ordered List* method, *POL-filter*, is described below.

Similar to COL-filter, we maintain inverted lists for each term t_i, denoted by l_i with entries $\langle p.id, p.v_i \rangle$ as defined above. For each list l_i we consider r groups which we identify by their boundaries: $b_{i1} > ... > b_{ir}$. The entries in l_i are grouped based on these boundaries. An entry $\langle p.id, p.v_i \rangle$ belongs to the group b_{ij} if $p.v_i \geq b_{ij}$ and $p.v_i < b_{i(j-1)}$, where the second condition is assessed only for $j > 1$. The entries inside one group are not kept sorted. To process an incoming document $d = (w_1, ...w_m)$, we start with the first group b_{i0} in all lists l_i where $w_i > 0$ and calculate the similarity scores of profiles in these groups with regard to d. The complete weight vector of a profile can be known, when necessary, by accessing the profile hash table in constant time. A profile's result set is updated with d when $sim(d, p) > p.s$. We continue to assess the profiles in the next group in each list only if the following stopping condition is not satisfied:

$$g(f_{w_1}(b_{10}), ...f_{w_m}(b_{m0})) < 1$$

In the following rounds, the algorithm reads all profiles which appear in group $b_{j(t+1)}$ where j identifies the list with the largest $w_j b_{j(t)}$ value and t is the last group assessed in list j. The stopping condition, replacing $b_{j(t)}$ with $b_{j(t+1)}$ in the above equation, is assessed each time a new group $b_{j(t+1)}$ is processed. It is easy to see, similar to the proof of Theorem 1, that this procedure is complete. Note that similar to COL-filter we can use the fact that the number of terms per profile is much smaller than the number of lists which are under process and improve the stopping condition.

The update cost in POL-filter is limited to maintaining the groups in each list. Since the entries in a group are not sorted, a hash table which provides constant insert and removal costs could be used to add or remove entries to groups. We move a profile p from a group when $p.s$ changes *and* the group membership does not hold anymore for the current group. In this case, the algorithm identifies the newly qualified group and the two affected groups are updated. Note that $p.s$ can change due to arrival of a new document which qualifies as p's top-k result or expiration of a previously top-k document.

4.2 Boundary Selection

While POL-filter decreases the maintenance cost, its effectiveness on early stopping depends on the selected group boundaries. If the b_i values are chosen to be too big, the stopping condition is not satisfied and the algorithm processes all the profiles in a list. On the other hand if they are chosen to be too small, the algorithm may process too many unnecessary profiles before it stops. Fixing the number of groups to r, we measure the extra cost POL-filter incurs by processing unnecessary profiles and aim at minimizing it. To calculate this extra cost, we assume that the lists are sorted also inside the groups, and calculate the number of extra profiles processed in a group. For simplicity let us assume we have a single term t and its corresponding list l. We later show how our discussion is extended to multiple lists. We denote the weight of t in an incoming document with w and the entries in l by $p.v$. We consider r groups with boundaries $b_1 > ... > b_r$. In case of a single list, if group b_i's profiles have been assessed, the POL-filter's stopping condition is $g(f_w(b_i)) < 1$. Let W and V denote the random variables corresponding to w, weight of t, and v values. Assume $f_W(x)$ and $f_V(x)$ respectively show the probability distribution functions of W and V. Also assume the size of list l is q and the number of documents with term t is n. The following is the cost which POL-filter incurs:

$$\sum_{i=1}^{r} n \int_{b_i}^{b_{i-1}} f_V(x) \int_0^1 f_W(z/x) q Pr(b_i < V < x) \delta z \delta x$$

where b_0 is the largest value V can have and b_r the smallest value. Since we have assumed the profiles are sorted also in the groups, $n \int_{b_i}^{b_{i-1}} f_V(x) \int_{\infty}^{1} f_W(z/x) \delta z$ counts the number of times the stopping condition is satisfied for a profile in group b_{i-1} and $qPr(b_i < V < x)$ counts the number of extra profiles POL-filter reads in this case. Since POL-filter checks the stopping condition each time a new group is assessed, it reads at most "*size of a group*" extra profiles compared to COL-filter. The above simplifies to:

$$\int_{b_0}^{b_r} nqx f_V(x) Pr(W < 1/x) Pr(x) \delta x$$

$$-\sum_{i=1}^{r-1} Pr(b_i) \int_{b_i}^{b_{i-1}} x f_V(x) Pr(W < 1/x) \delta x$$

b_0 and b_r are fixed values (maximum and minimum values of V). So to minimize the cost, the negative part should be maximized. While this is easy for some distributions like the uniform distribution, it is not straight forward for others. In our experiments we estimate the distributions of interest by histograms and solve the above optimization problem numerically.

In deriving the previous optimization equations, we assumed that only one list is under process. However, in a real scenario often several lists are being processed and the stopping condition depends on all of them. Therefore $g(f_{w_j}(b_{ji})) < 1$ is not a good estimate of the stopping condition. We treat the lists independently and use the maximum number of terms a profile can have (m') to estimate the stopping condition by $g(f_{w_j}(b_{ji})) < 1/m'$ instead. To account for this change in the above equations, $Pr(W < 1/x)$ should be replaced with $Pr(W < 1/xm')$.

5. RESULT MAINTENANCE

So far we have considered the problem of efficient profile filtering when a new document arrives. Another important aspect of our streaming scenario is the sliding window which specifies the valid documents. In this section we first describe the challenges caused by this temporal factor and then describe our solution.

When a document which is part of the top-k results of a profile expires as the window slides, another document from the existing valid documents should replace it. The process of re-evaluating a top-k query usually incurs high cost on the system. Given the fact that we aim at supporting a large number of profiles, this cost can slow down the system and prevent it from timely and correct responses to the users. This problem is closely related to view maintenance discussed in the database community [29].

In the following, we consider a given profile p and when we mention the score of a document or the top-k results, it is with regard to this specific profile. Let us assume a set $p.R$ of documents is maintained for p. To avoid ever re-evaluating p over the set of *all* valid documents, $p.R$ should contain all documents which have a chance of becoming a top-k result in their life time. This set consists of the top-k current results as well as documents which have a smaller score or shorter life time compared to at most $k-1$ other documents. This concept has been previously exploited in the context of continuous top-k processing in [20] and in kNN queries in [6]. For a given profile p, we consider the documents in the time/score space where score corresponds to the similarity degree of a document and p, and *time* presents its arrival time. A document d_1 dominates d_2 if $d_1.time > d_2.time$ and $d_1.score > d_2.score$. The k-skyband [20] of a set of points is a subset of these points where each is dominated by at most $k-1$ other points. Clearly if a document is not in the k-skyband it can never be a top-k result of p, as at least k documents with higher similarity grades and longer life times exist.

While many new documents do not qualify as relevant results to a profile due to their low similarity degrees, they are part of the k-skyband as a result of their *time* dimension: since we are considering *in order* streams, *all* incoming documents are part of the k-skyband of *all* profiles at the time they arrive. Such documents remain in a profile's k-skyband only for a short amount of time until they are dominated by fresher, more relevant documents. Inserting each incoming document to all profiles' k-skybands, incurs a large space overhead as well as unnecessary CPU cost to actually maintain the k-skyband.

Algorithm 2: The overall Algorithm for removing expired documents and inserting new documents

```
Input: $newdocs$,$expireddocs$
foreach document $d \in expireddocs$ do
    $D$.remove($d$);
    foreach profile $p \in d.profiles$ do
        $p.R$.remove($d$);
        if $|p.R| < k$ then
            re-evaluate($p$);
            $p$.updateTopK($p.R$.topK);
$tobeUpdated = \emptyset$;
foreach document $d \in newdocs$ do
    D.insert(d);
    $tobeUpdated$ = ProfileFilter($d$);
    foreach profile $p \in tobeUpdated$ do
        $p.R$.insert($d$);
        $p.R$.updateDominance();
        if $p.R.topK$ has changed then
            $p$.updateTopK($p.R$.topK);
```

To circumvent these costs, we restrict the documents which are inserted to $p.R$ to those which have scores larger than a threshold τ and maintain the k-skyband over them. We call this part of the k-skyband the ***horizon***. With suitable values of τ, the horizon is expected to be more stable, i.e. its members do not disqualify frequently, and more promising, i.e. its members are more likely to actually become a top-k result. In the following we first describe our *horizon result maintenance* method and then discuss suitable values of τ.

Consider a profile p and its corresponding set of documents $p.R$. A re-evaluation over the set of all valid documents is invoked when $|p.R| < k$ and in this case $p.R$ is set equal to the obtained top-k results. A newly arrived document d is inserted to $p.R$ if $sim(d,p) \geq \tau$. When a new document is inserted in $p.R$ the dominance values (i.e., a counter) of existing documents are updated accordingly and those documents whose dominance value hits k are eliminated from $p.R$. Note that removing a document from $p.R$, either due to expiration or as a result of dominance by k other documents, does not affect the dominance values of other existing documents: all documents dominated by this document should have been removed before.

Fixing τ to a predefined static value is not suitable for our dynamic setting as an appropriate value currently may be too small or big in future with regard to the corresponding window of valid documents. A too small value would result in all documents qualifying for insertion to $p.R$ and ultimately maintaining the complete k-skyband. On the other hand, a too large value causes $p.R$ to frequently contain less than k documents and firing numerous re-evaluations. A dynamic value for τ which adapts to the relevance of current documents is the remedy.

Let *p.R.score* denote the score of the ranked last document in $p.R$. We show that for any value of τ smaller than or equal to *p.R.score*, $p.R$ contains the correct top-k results: the top-k results in $p.R$ are the same as the result of evaluating p over all valid documents. We also show, by a contradicting example that this is the largest value which still guarantees the correctness of the horizon. Note that the *correctness* concern raises due to dynamically changing τ.

THEOREM 2. *Let p.R.score denote the similarity score of the ranked last document in p.R. If τ is changing dynamically, the necessary and sufficient condition for p.R to contain the correct top-k results is that $\tau \leq p.R.score$.*

For the proof please see the appendix.

5.1 Integration with Profile Filtering

To integrate the above described horizon maintenance scheme with the proposed profile filtering algorithm, the $p.s$ values used in calculating $p.v$ in Section 4 should be replaced with τ. This will ensure that the profile filtering algorithm will not miss any profiles p where a new document should be inserted to $p.R$, although the document may not qualify as a top-k result of p currently. The overall steps for inserting documents and updating profiles are shown in Algorithm 2. D denotes the set of valid documents. First, expired documents are removed from D. $d.profiles$ denotes the affected profiles by d: all profiles p where $d \in p.R$. An affected profile p of an expired document is re-evaluated if $|p.R| < k$. Then, for each of the incoming new documents the profile filter returns the profiles which should be updated with this document. Note that if document d is inserted in $p.R$ it is not necessarily a top-k result of p, but it is part of p's horizon.

6. EXPERIMENTS

We have implemented a simulation of the envisioned system in Java 1.6. The dataset is stored in an Oracle 11g database and replayed according to the timestamps attached to the entries.

Dataset and Profiles

We have obtained the ICWSM 2009 Spinn3r Blog Dataset[3]. It consists of 44 million blog posts between the time period of August 1st and October 1st, 2008. Each blog entry (post) consists of plain text, a timestamp, a set of tags, and other meta information such as the blog's homepage URL. The data is formatted in XML and is further arranged into tiers approximating to some degree search engine ranking. We have parsed the blog posts for the highest tier levels resulting in $2,444,780$ distinct posts. After stemming and stopword removal, we have inserted the content of each blog as $\langle term, score\rangle$ pairs in the database where the TF/IDF methodology is used for assigning weights.

Profiles are generated by looking at frequently used topic descriptions of the blog entries, such as "US election". Each profile has between 3 and 5 out of 657 distinct terms and their corresponding weights are chosen uniformly at random. We did not use one of the standard search engine query logs as subscription queries are of a more general nature. We use the well-accepted cosine measure to calculate the similarity degree between a document and profile. Note that as mentioned in Section 1.1 the cosine measure has the two necessary properties of monotonicity and homogeneity. We assume that the document and profile vectors are normalized by their lengths: $|p| = |d| = 1$, so $sim(d, p) = \sum_{i=1}^{m} w_i u_i$.

Algorithms and Measures of Interest

We consider the following three algorithms for profile filtering.

[3]http://www.icwsm.org/2009/data/

Parameter	Default	Range
number of profiles	50K	30,40,50,60,70,80
result cardinality (k)	10	5,10,15,20
window size	4500(s)	2700,3600,4500,5400,6300
number of groups	10	2,4,6,8,10,12,14

Table 1: Variations of the parameters as used in the experiments.

Algorithms	total	profile fil.	update	re-eval	insert
naive-k	23.12	7.45	0.00	14.73	0.67
col-k	23.87	2.36	3.12	17.39	0.74
pol-k	21.14	3.48	0.52	16.17	0.71
naive-horizon	11.01	7.18	0.00	1.71	1.80
col-horizon	9.64	3.47	1.30	2.71	1.83
pol-horizon	8.84	4.60	0.28	1.78	1.85
incr.-thresh	13.86	-	-	-	-

Table 2: Average time measurements (ms).

- **naive:** As a baseline we have implemented a profile filtering algorithm that keeps non-sorted inverted lists of profiles and reads, for an incoming document, all entries from all inverted lists that correspond to a term in the document.

- **col:** This is our algorithm as described in Section 4 which keeps all profiles in term-based index lists, sorted by score. The exact ordering is kept at all times, which has benefits for the profile filtering process but comes with the cost of placing or re-placing entries to the exact position w.r.t. their scores.

- **pol:** This algorithm as described in Section 4.1 divides the inverted lists to groups, maintaining the group membership criteria for the entries but not the order among entries of a particular group. We expect a larger percentage of profiles read during the profile filtering but a significantly lower maintenance cost.

For all the above we have two alternatives for result maintenance: (i) using a simple top-k list and re-evaluating whenever one of the top-k results expires or (ii) maintaining each profile's *horizon*, as explained in Section 5.
We have also implemented **the approach by Mouratidis et al.** in [21], described shortly in Section 2, which we will refer to it as **incr.-thresh**.

We report on CPU time as our main measure of performance. Note that we do not report on accuracy measures as all algorithms report the exact top-k results. To better understand the effects of our proposed algorithms we have measured CPU time for different parts of the algorithms, in addition to the overall time. Also for the result maintenance algorithm we are interested in the space overhead imposed by retaining more than k documents per profile.

Depending on the sliding window size, the number of documents inserted in the system, ordered on their timestamps, is such that at least 5 non-overlapping sliding windows have completed. All measurements are averaged over all processed documents after a warm-up phase of 500 documents. The parameters, their default values and range of variations are shown in Table 1. All algorithms are run on a quad-core Intel Xeon CPU E5530 @2.4 GHz, 48 GB main memory, 4 TB of hard drive and Microsoft Windows Server 2008 R2 x64 as operating system.

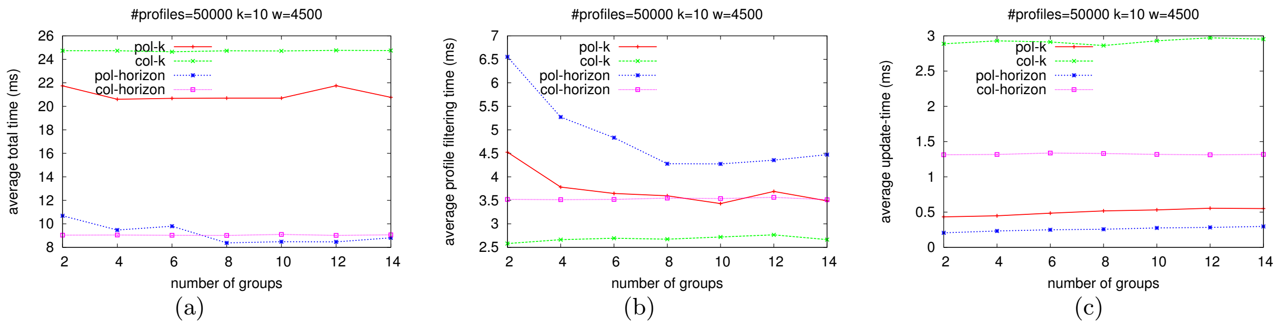

Figure 2: The effect of number of groups (a) average total time (b) average profile-filtering time (c) average update time.

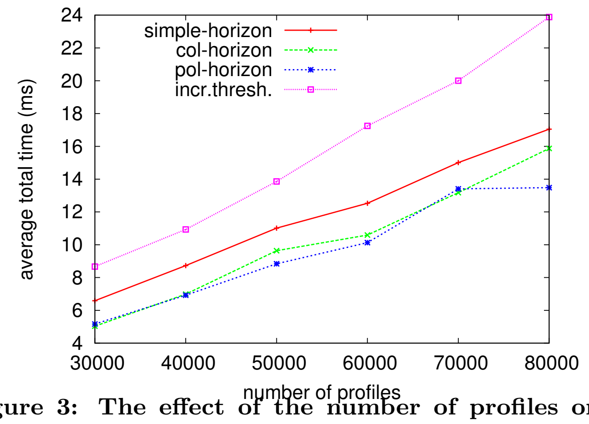

Figure 3: The effect of the number of profiles on average total time.

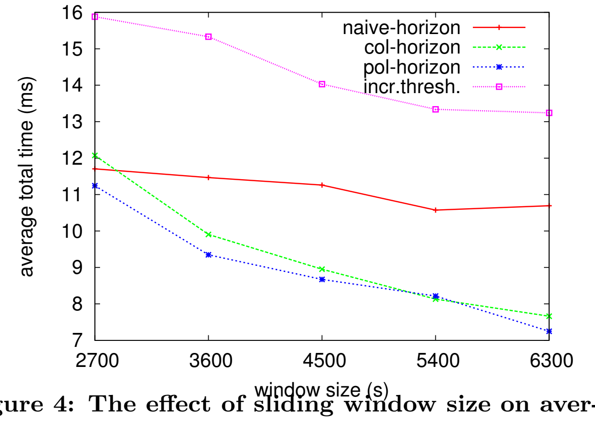

Figure 4: The effect of sliding window size on average total time.

Results

Table 2 shows the time measurements for the default system parameters for naive, col, pol and, incr.-thresh. Total time includes the time spent for removing old documents, updating the affected profiles by re-evaluating them, profile filtering for a new incoming algorithm and inserting it to the result sets of selected profiles. To have a better understanding of the effect of different approaches, Table 2 shows the time spent for each of these parts separately. Note that we do not show the time which is the same for all algorithms, like the time to insert a document in the term-document inverted list, but this is included in the total time. Furthermore, for incr.-thresh we only show the total time, as this algorithm does not have same separated modules for the above mentioned tasks. The first observation is that a significant portion of time (31%) is spent in the profile filtering component in the naive-k algorithm. The col-k algorithm decreases this time by almost 68% at the expense of large update time for keeping its necessary structures up-to-date. Our proposed pol-k algorithm is successful in decreasing the time spent for profile filtering as well as limiting the update cost. Note that col-k has a bigger re-evaluation time, as re-evaluating a profile causes its *p.s* value to change, causing updates in the inverted lists which should be kept sorted for col-k. While col-k incurs a larger total time due to its huge update cost, pol-k achieves in total 8% improvement compared to naive-k. The next three rows of this table show the measurements for the horizon variation of the algorithms. The re-evaluation cost decreases for all algorithms by almost 84% at the expense of a relatively small increase in the result insertion time. In the horizon variations, result insertion is more costly as it involves updating the dominance counters and k-skyband maintenance. Since with the horizon method more profiles get qualified to have a document in their result set, we observe an increase in the profile filtering time for col-horizon and pol-horizon compared to their top-k counterparts. However, since the ranked last document in the result set, which defines the values of interest for keeping the lists ordered, changes less frequently than in the top-k method, the update cost for these algorithm decreases significantly. In total, we observe 60% decrease in the total time, from naive-k to pol-horizon which achieves the smallest total time among all algorithms. incr.-thresh inserts all documents that are in any index list above the scan line of that profile which causes the result set to grow very large. This has the benefit of eliminating the re-evaluations, but on the downside large space is consumed and a large result set should be kept sorted which incurs extra cost. Overall, pol-horizon has a decrease in total time of 36% together with significant decrease in the resultset size it maintains compared to incr.-thresh.

As seen previously, the pol algorithm is successful in maintaining the decrease in profile filtering time as well as limiting the time spent for updating the required structures. The calculations necessary for choosing the boundary values mentioned in Section 4.2 are performed only once after the warm-up phase. Figure 2 presents the effect of number of groups. Figure 2 (a) shows the total time for pol and col with the top-k and horizon variations. We observe that with as small as 10 groups, pol achieves very good decrease in the total time. In Figures 2 (b) and (c) we observer the profile filtering and update times for different number of groups. pol-k incurs almost 6 times less update cost compared to col-k, at the expense of small increase in its profile filtering time. As mentioned in the previous paragraph, the hori-

k	#reeval top-k	size top-k	#reevals horizon	size horizon
5	73.18	4.99	11.49	7.93
10	141.03	9.99	6.56	17.84
15	209.00	14.97	3.55	28.75
20	278.51	19.94	2.62	40.11

Table 3: Number of re-evaluations and result set sizes when changing k. w=4500(ms) and #profiles = 20000.

zon variations have smaller update cost and slightly larger profile filtering time.

The effect of number of profiles on total time is shown in Figure 3. The total time for all algorithms increases with increasing the number of profiles, as the profile filtering and re-evaluation parts become more costly. However, the effect of our profile filtering algorithms are more visible for larger number of profiles. Also note that pol does not show any significant drop in decreasing the total time compared to col, although the number of groups are fixed for all profile cardinalities to 10. This is because by using the horizon maintenance module, the update cost in col decreases significantly, as shown in Figure 2(c). incr.-thresh has much larger total time, since similar to col-k it spends a lot of time updating the index lists' scan lines. The pol-horizon algorithm achieves the minimum total time, decreasing it by up to 40% from incr.-thresh

We report on the effect of sliding window size on average total time in Figure 4. The average total time decreases with increasing size of the sliding window for all algorithms. This is mainly due to the decrease in number of necessary re-evaluations on average. With a bigger window size, high quality documents live longer and a larger time span allows for high quality documents to arrive before others expire and fire a re-evaluation. As seen in the Figure, with large enough window sizes pol-horizon and col-horizon have similar total time which is the result of fewer updates.

Table 3 reports on the average number of re-evaluations and result set size for the top-k and horizon variations. Note that the profile filtering algorithm does not have an effect on these values so we have not repeated the results by separately reporting on them. First, note that the necessary number of re-evaluations drops from 7 to almost 100 times less for the horizon method compared to top-k based maintenance. The very interesting observation is that with increasing the k value, the number of re-evaluations has an increasing trend for the top-k method but a decreasing one for the horizon algorithm. This is because for larger values of k the horizon grows much larger than k, significantly decreasing the chance of the result set containing less than k results to fire a re-evaluation. However, the horizon method comes with the extra cost of maintaining the horizon.

In summary we observe that the pol-horizon combination offers significant performance gains compared to the rest of algorithms. The horizon result maintenance algorithm causes small decrease in the improvement pol can offer in decreasing the profile filtering time. However, it drastically decreases the necessary update cost and number of re-evaluations while incurring only a small space over head over the system. A decrease of between 25% to 30% in overall processing time, allows our envisioned system to scale better to larger number of profiles and higher data rates.

7. CONCLUSION

Motivated by the tremendous popularity of blogs, microblogging services like Twitter, and online newspapers, we address the problem of continuously processing large number of user defined subscription queries (profiles) over a stream of documents. The challenge in processing these queries in real-time lies not only in the fact that there are many queries, but also, and foremost, in the observation that data streams in at high rates. Both properties combined call for a careful profile filtering process, that omits evaluating too many profiles. Our approach significantly reduces the number of necessary profiles evaluations, by organizing the user profiles in a so called inverted index. The key idea is to sort profiles not only on their weight w.r.t. a term but also according to the quality of the currently alive documents which are ranked high for a particular profile. This sorting criteria allows for an effective stopping condition for the profile filtering algorithm. Furthermore, we observe that keeping the entire lists completely sorted is infeasible as profiles frequently move up and down in the lists. We solve this by using group sorted lists, i.e., lists consisting of different groups which are sorted relative to one another, but without order inside groups. As the definition of the group boundaries is crucial for the overall performance gain, we present a method to select these bounds by leveraging score distribution information derived from histograms. We combine our proposed filtering algorithm with an effective skyline based result maintenance algorithm which cuts drastically on the number of necessary re-evaluations caused by expiring documents. We evaluate our approach using a real world blog dataset demonstrating the performance gains compared to the state-of-the-art.

8. REFERENCES

[1] Xiang Wang, Kai Zhang, Xiaoming Jin, and Dou Shen. Mining common topics from multiple asynchronous text streams. *WSDM*, 2009.

[2] James Allan, Jaime Carbonell, George Doddington, Jonathan Yamron, and Yiming Yang. Topic detection and tracking pilot study final report, 1998.

[3] James Allan, Ron Papka, and Victor Lavrenko. On-line new event detection and tracking. *SIGIR*, 1998.

[4] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. *PODS*, 2002.

[5] Timothy A. H. Bell and Alistair Moffat. The design of a high performance information filtering system. *SIGIR*, 1996.

[6] Christian Böhm, Beng Chin Ooi, Claudia Plant, and Ying Yan. Efficiently processing continuous k-nn queries on data streams. *ICDE*, 2007.

[7] Stephan Börzsönyi, Donald Kossmann, and Konrad Stocker. The skyline operator. *ICDE*, 2001.

[8] James P. Callan. Document filtering with inference networks. *SIGIR*, 1996.

[9] Gautam Das, Dimitrios Gunopulos, Nick Koudas, and Nikos Sarkas. Ad-hoc top-k query answering for data streams. *VLDB*, 2007.

[10] Ronald Fagin. Combining fuzzy information: an overview. *SIGMOD Record*, 31(2), 2002.

[11] Parisa Haghani, Sebastian Michel, and Karl Aberer. Evaluating top-k queries over incomplete data streams. *CIKM*, 2009.

[12] Qi He, Kuiyu Chang, and Ee-Peng Lim. Analyzing feature trajectories for event detection. *SIGIR*, 2007.

[13] Andreas Hotho, Robert Jäschke, Christoph Schmitz,

and Gerd Stumme. Trend detection in folksonomies. *SAMT*, 2006.

[14] Cheqing Jin, Ke Yi, Lei Chen 0002, Jeffrey Xu Yu, and Xuemin Lin. Sliding-window top-k queries on uncertain streams. *PVLDB*, 1(1), 2008.

[15] Jon Kleinberg. Temporal dynamics of on-line information streams. *In Data Stream Management: Processing High-Speed Data*, Springer, 2006.

[16] Flip Korn and S. Muthukrishnan. Influence sets based on reverse nearest neighbor queries. *SIGMOD*, 2000.

[17] Nick Koudas, Beng Chin Ooi, Kian-Lee Tan, and Rui Zhang 0003. Approximate nn queries on streams with guaranteed error/performance bounds. *VLDB*, 2004.

[18] Ravi Kumar, Jasmine Novak, Prabhakar Raghavan, and Andrew Tomkins. On the bursty evolution of blogspace. *World Wide Web*, 8(2), 2005.

[19] Michael Mathioudakis and Nick Koudas. Efficient identification of starters and followers in social media. *EDBT*, 2009.

[20] Kyriakos Mouratidis, Spiridon Bakiras, and Dimitris Papadias. Continuous monitoring of top-k queries over sliding windows. *SIGMOD*, 2006.

[21] Kyriakos Mouratidis and HweeHwa Pang. An incremental threshold method for continuous text search queries. *ICDE*, 2009.

[22] S. Muthukrishnan. Data streams: Algorithms and applications. *Foundations and Trends in Theoretical Computer Science*, 1(2), 2005.

[23] Amit Singh, Hakan Ferhatosmanoglu, and Ali Saman Tosun. High dimensional reverse nearest neighbor queries. *CIKM*, 2003.

[24] Ioana Stanoi, Divyakant Agrawal, and Amr El Abbadi. Reverse nearest neighbor queries for dynamic databases. *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, 2000.

[25] Christos Tryfonopoulos, Manolis Koubarakis, and Yannis Drougas. Information filtering and query indexing for an information retrieval model. *ACM Trans. Inf. Syst.*, 27(2), 2009.

[26] Roger Weber, Hans-Jörg Schek, and Stephen Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. *VLDB*, 1998.

[27] Tak W. Yan and Hector Garcia-Molina. Index structures for information filtering under the vector space model. *ICDE*, 1994.

[28] Tak W. Yan and Hector Garcia-Molina. Index structures for selective dissemination of information under the boolean model. *ACM Trans. Database Syst.*, 19(2), 1994.

[29] Ke Yi, Hai Yu, Jun Yang, Gangqiang Xia, and Yuguo Chen. Efficient maintenance of materialized top-k views. *ICDE*, 2003.

APPENDIX

PROOF 1. We show that for any profile which has not been updated before the stopping condition is reached, d does not serve as a top-k result. In other words we show that for such profiles, $sim(d,p) < p.s$. If p has been seen in one of the sorted lists before the stopping condition, according to the algorithm its similarity score with d has been evaluated by looking up p in the profile hash table. Therefore if p has not been updated, clearly $sim(d,p) < p.s$. Now assume p has not been observed in any of the sorted lists before the stopping condition. For a list l_i let $\underline{v_i}$ be the last observed value under sorted access. Since the lists are sorted in descending values, $p.v_i < \underline{v_i}$. As a result of this and due to f and g's monotonicity, $g(f_{w_1}(v_1), ...f_{w_m}(v_m)) \leq g(f_{w_1}(\underline{v_1}), ...f_{w_m}(\underline{v_m})) < 1$ where the last equality is the stopping criteria. Since $p.v_i = p.u_i/p.s$ and due to f and g's homogeneity, we have

$$g(f_{w_1}(v_1), ...f_{w_m}(v_m)) = g(f_{w_1}(u_1)/p.s, ...f_{w_m}(u_m)/p.s)$$

$$= g(f_{w_1}(u_1), ...f_{w_m}(u_m))/p.s < 1$$

which is equivalent to $sim(d,p) < p.s$. □

PROOF 2. We first show that if $\tau \leq p.R.score$, $p.R$ contains the true top-k. Let d be the valid document with the largest score which is not in $p.R$ at current time $t_{current}$ and $p.R.score_k$ be the score of the ranked k document in $p.R$ also at $t_{current}$. We show that $sim(d,p) < R.score_k$. We should consider two cases: first d was inserted to $p.R$ but then removed, or d was never inserted to $p.R$. Since d is valid, it was removed from $p.R$ as a result of being dominated by k documents which means k documents with longer life times exist which have a higher score than d. These are indeed in $p.R$, as we have assumed d has the largest score among all valid documents *not* in $p.R$. So for the first case $sim(d,p) < p.R.score_k e$. In the second case, d was never inserted to $p.R$. Let t_1 denote the time when the most recent re-evaluation was performed. If $t_1 > d.time$, the most recent re-evaluation was performed after d's arrival. Since d was not inserted in $p.R$, at least k documents with higher scores than d existed at time t_1. Since $|p.R| = k$ after each re-evaluation, τ at after t_1 and before a new re-evaluation is equal to or larger than the score of the ranked k document at time t_1. Since we have assumed the most recent re-evaluation happened in t_1, either those top-k documents have not expired until $t_{current}$, or documents with score larger than τ_{t_1} have arrived, otherwise the size of $p.R$ would be less than k at some point after t_1 which is in contradiction with our assumption that the most recent re-evaluation was invoked at t_1. In both cases $R.score_k \leq \tau_{t_1} > sim(d,p)$. Now assume $t_1 < d.time$: the most recent re-evaluation happened before d's arrival. In this case, $sim(d,p) < \tau_{d.time}$, otherwise d was inserted in $p.R$. If no re-evaluations happens, τ can only increase, as only higher scored documents can be inserted to $p.R$. Similar to the previous case, either documents in $p.R$ at time $d.time$ have not expired yet or higher scored documents have arrived, otherwise a re-evaluation would have been fired. In both cases, $R.score_k \geq \tau_{d.time} > sim(d,p)$ which completes the proof for correctness of results when $\tau \leq R.score$.

To show that this is also a necessary condition, we give an example of when $p.R$ doesn't contain top-k results if $\tau < R.score$. For simplicity let $k = 2$, examples for other k can be constructed similarly. Let $\tau = R.score + \epsilon$ and $\epsilon > 0$. Assume R is empty and consider the following stream of documents (first attribute shows time and the second is score with regard to the specific profile we consider): $d_1(1, s_1)$, $d_2(2, s_2)$, $d_3(3, s_3)$, $d_4(4, s_4)$, where $s_1 > s_2$, $s_1 > s_3$, $s_3 > s_2$. Then when d_4 arrives, $\tau = s_2 + \epsilon$, because d_2 is dominated only by d_3 so it isn't removed. Now if $s_4 = s_2 + \epsilon/2$, d_4 is not inserted to R. Assume no new document arrives. When d_1 expires, d_2 and d_3 are reported as the top-k results although d_4 has higher score than d_2. □