Comput Sci Res Dev (2012) 27:81–92
DOI 10.1007/s00450-011-0178-9

SPECIAL ISSUE PAPER

Efficient monitoring of personalized hot news over Web 2.0 streams

Parisa Haghani · Sebastian Michel · Karl Aberer

Published online: 21 May 2011

Abstract Web 2.0 streams, like blog postings, micro-blogging tweets, or RSS feeds from online communities, offer a wealth of latest news about real-world events and societal discussion. From a user's perspective, it becomes harder and harder to get a decent overview of recent events, given these massive streams of information that are continuously flowing. Ideally, a system would continuously put together recent information, ranked by the current social impact but also weighted by the users' personal interests. In this work, we develop methods to meet these requirements. The presented approach continuously tracks the most popular tags attached to the incoming items and based on this, constructs a dynamic top-*k* query. By continuous evaluation of this query on the incoming stream, we are able to retrieve the currently *hottest* items. These hottest items are then fed into an engine that re-ranks them w.r.t. user specified interests, given in form of term based topic descriptions. This calls for high performance algorithms for efficient hot document retrieval and subsequently personalizing these documents based on user profiles, given the high rate of incoming data and the immense number of user profiles. In this work we present a combined solution, making use of our prior work on information filtering and showing how it can be used in combination with the current work, on how to continuously determine the hottest documents. To demonstrate the suitability of our approach, we perform a performance evaluation using a real-world dataset obtained from a weblog crawl.

Keywords Web 2.0 · Stream processing · Personalization

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).

P. Haghani · K. Aberer
EPFL IC ISC LSIR, Station 14, 1015 Lausanne, Switzerland

P. Haghani
e-mail: parisa.haghani@epfl.ch

K. Aberer
e-mail: karl.aberer@epfl.ch

S. Michel (✉)
Saarland University, Campus El.7, 66123 Saarbrücken, Germany
e-mail: smichel@mmci.uni-saarland.de

1 Introduction

The world has turned into one large-scale interconnected information system with millions of users. End users, with the advent of Web 2.0, are now content generators who actively contribute to the Web. User generated data is usually in form of semi-structured text like personal blog entries with categorization[1] or images and videos annotated with tags [14, 32]. Information is flowing continuously and at an immense rate, calling for efficient and effective means to extract the essence of the available information.

As an *example* of temporal streams of information in a Web 2.0 application consider published content in form of news articles or posts on personal weblogs (blogs). Explicit temporal annotations (i.e. *written at*, *uploaded at*) of the content of weblogs or news portals makes them natural items of a temporal stream. Mechanisms such as RSS and atom are used to notify users of newly published data on their favored weblogs or news portals. The items in a blog feed stream are

[1] http://google.blogspace.com/, http://www.weblogs.com/, http://www.blogger.com/.

generated at distributed sources depending on the subscriptions which are made by the user. The large body of information retrieval techniques can be used in order to extract categories or topics from the published text [1, 2]. Data stream processing has gained a lot of attention in the recent years (see [5, 29] for surveys), since many of today's applications are best captured in this model. Data items in different formats stream in to a processing unit where each item has the chance of being seen once before being archived for later uses. Given the immense volume of data being published on the web and the desire of consuming newly published data, there is an increasing need for processing this information in real time in efficient ways. All this gives rise to considering this data in a streaming model.

Our approach considers extracting the most popular documents out of the streaming data. It is based on detecting the most popular (hottest) tags used in the recent past, defined by a sliding window over the stream, treating these tags as a top-k query over the stream of recent documents. As the set of hot tags changes with time, our considered quarry changes as well. Therefore, our problem is different from evaluating a fixed query over a stream of incoming documents, which is a well addressed problem [28]. Furthermore, while this model reflects the recency and popularity of the retrieved documents it does not immediately cover particular user interests. We show how recent advances in the area of personalized information filtering can be applied in combination with the engine that retrieves the hottest documents. This paper is based on our earlier work in [16].

1.1 Problem statement and contribution

We consider a stream of tagged items where each item has the following format:

$$d = \langle itemId, time, \mathcal{T}_d \rangle$$

itemId is a unique identifier specifying the object this item is describing, i.e. URL of an image or post, and *time* represents the time when d was produced. Let $\mathcal{T} = \{t_1, \ldots, t_n\}$ be the global set of tags which are used to annotate items. $\mathcal{T}_d \subset \mathcal{T}$ is the set of tags with which d is annotated. The number of tags an item carries is usually very small (e.g., around 5) compared to standard document retrieval where a text document contains lots of terms. We assume each tag $t \in \mathcal{T}_d$ is associated with a normalized (in [0, 1]) score $score(d, t)$ that reflects the relatedness of t to the item d.

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.

At each point in time we can compute the usage frequencies of tags occurring in documents currently in the sliding window W or compute aggregation queries over the documents themselves. This view forms the basis of our approach, which builds on statistics on tag usages to determine a set of popular tags. This tag set is then interpreted as a continuous and dynamic keyword query which is executed against the sliding window as time evolves. We call this query dynamic as it is re-build with evolving time due to changes in tag usage frequencies.

Definition 1 (Hot Tags and Hot Items) At each timestamp τ, the set of hot tags (H_τ) consists of the c tags with the highest popularity (frequency) in the current sliding window.

The set of hot tags defines the query we use to rank the valid items, i.e., the query is data-driven and changes with time as the popularity of tags changes. For a valid item, we define its current score as the sum of scores of the hot tags it carries. More formally,

$$s(d, H_\tau) := \sum_{t \in H_\tau \cap T_d} score(d, t) \tag{1}$$

The task is to continuously compute the top-k items as the query changes. In contrast to standard top-k query processing over text (or XML) documents, here, the query is supposed to be rather big to capture not only a few but many hot topics for diversity reasons. In summary, the considered tags (features, in standard IR terminology) is small whereas the query is long, which is in clear contrast to traditional query processing techniques.

In this work we focus on efficiency aspects and the potential of pre-aggregations and how to decide which subqueries to pre-compute. For the actual decision which tags should be considered in as query terms, one can think of other measures than the pure popularity count based methods we use in our work, e.g., methods that aim at identifying trending (hot) topics (cf., e.g., [4, 25]).

Figure 1 reports on the overall architecture: the blue box on the right depicts the hot k document tracking component, whose output it fed into the personalization engine which displays results immediately to the registered users. In this paper we make the following contributions. We show how to continuously compute the set of hot items over social (Web 2.0) data streams by defining a dynamic top-k aggregation query and show how pre-aggregations of popular sub queries can be used to efficiently process the query. Additionally, we describe a viable solution for furthermore personalizing these hot items based on user specified preferences. We evaluate the hot item tracking methods on a real-

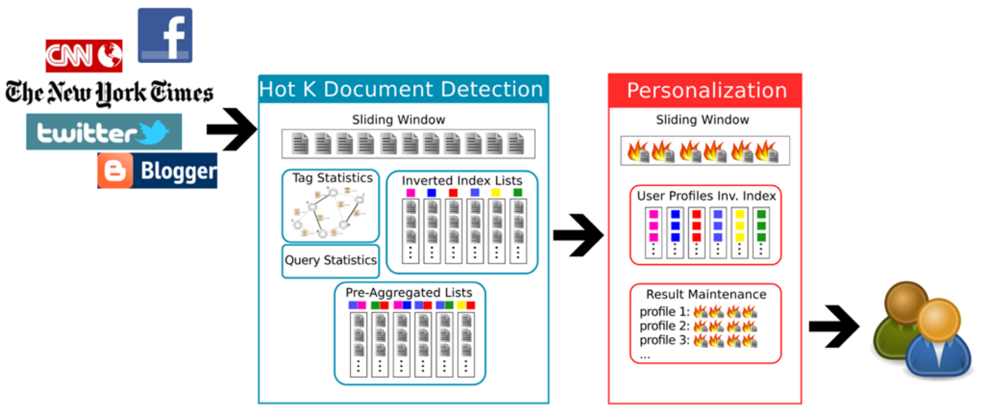

Fig. 1 The Overall Architecture of our approach. (The logos are trademarks of the corresponding companies)

world dataset of blog posts showing the suitability of our approach.

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 the problem of pre-aggregating groups of index lists for efficient query processing and presents next to an offline problem definition an efficient and effective approximation for online processing. Section 5 discusses the integration of user provided subscription queries. Section 6 presents the experimental evaluation. Section 7 concludes the paper.

2 Related work

Data stream processing has been a hot topic in the past years as many of today's applications require real-time processing of dynamic data. For comprehensible surveys of this topic in general see [5, 29]. Early works mostly consider one-pass algorithms in limited space over the whole stream where all tuples are considered valid at all times. A related problem to ours is reporting on *quantiles* or *heavy hitters* in streams. The goal is to report on most repeated items in the stream, when the number of items is so high that keeping statistics for each is not possible. Approximate solutions to this problem exist which make use of techniques such as the famous AMS sketches [3], or more recently group testing, see for example [8, 9] and the references within. In our work, the number of tags we consider and desire to know the hottest amongst is small enough such that exact statistics could be kept for each.

Another line of research in stream processing is dedicated to top-k query answering in data streams. Mouratidis et al. [28] maintain a skyline [6] 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. In a more general setting, [11] proposes indexing methods for answering *adhoc* top-k queries based on arrangements. While our queries can not be considered as *fixed* (as the set of hot tags changes over time with new items arriving) they are not completely *ad-hoc* either. We exploit this fact to pre-aggregate parts of the query which can be used several times in future queries. Jin et al. [20] consider top-k queries on uncertain streams where the data items are associated with existential probabilities. In our envisioned applications all items are certain.

Mainly motivated by the wealth of news feeds and other online information streams, another related problem is *Topic Detection and Tracking* (TDT) which has been extensively studied in the past few years [1, 2, 17]. The goal here is to detect new events appearing in the data stream and tracking those events in order to later identify data which further discuss the same event. Another related topic is mining frequent itemsets in a data stream. In a recent work Calders et al. [7] define a new measure as the frequency of an itemset and propose an incremental algorithm that allows for reporting the exact frequencies of frequent itemsets. The problem of itemset mining is orthogonal to our problem and can be used to improve the quality of our choice of pre-aggregation queries.

In another line of research related to Web 2.0 applications with temporal considerations, Hotho et al. [18] 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 based on the famous PageRank algorithm. They perform the algorithm in an offline manner and assume the whole corpus of data to be available. Weblog evolution is considered in [22], where *time graphs* are introduced and used for community tracking again in an offline mode. In [24], the goal is to identify weblogs defined as *starters* and *followers* specified by certain linking relations in an efficient way. In contrast to the above, we continuously evaluated the data as it arrives in an online manner. For a survey of temporal data analysis methods see [21] and the references within.

[2]http://del.icio.us.

Keeping the query results updated as data streams in with high rates requires high performance evaluation of top-k queries. One way to improve the performance of expensive queries is to maintain their results as materialized views. In order to avoid reprocessing a top-k query in face of updates in the database, such as insertions or deletions, authors in [31] suggest maintaining a top-k' view, where $k' > k$ and show how to choose k' dynamically to adapt to the system workload. In [19] authors investigate answering a top-k query based on the materialized results of another top-k query where the preference function is a linear combination of all attributes of tuples. It is shown how to decide given a preference function and it's top-l results if the top-1 result of another preference function can be found in these materialized l tuples. In [10], the TA algorithm is adapted to the case where a set of views, not necessarily the single inverted lists, are available. The views are visited in a lock-step manner and in each iteration the maximum score of unseen tuples are calculated by a linear programming optimization, given the preference functions of each of the views. Given a set of views, the best subset for answering a query is chosen based on a process simulating the TA utilizing the data distributions in each view. In the same line, [23], investigate top-k query processing when intersection of single inverted lists are also available. A combinatorial solution is proposed to solve the specific linear program appearing when the set of lists consist of only single or intersection of two single lists. A very interesting result of the paper is that in order to guarantee instance optimality all available lists should be investigated. In a streaming scenario however, maintaining the intersection of all pairs of single lists is not possible due to memory constraints. In this work we propose to maintain the intersection of several lists instead of just pairs of them and we chose the intersections based on the benefits they potentially have for future data-driven top-k queries.

3 System model and structure

In this section we briefly describe the general structure that we consider. As mentioned in Sect. 1.1 we consider one data stream as the input to our system where the items in this stream contain a list of tags and they are considered valid while belonging to a sliding window.

We assume all valid items are sorted in a first-in-first-out list. This provides an efficient mechanism for evicting expired items. Newly arriving items in the stream are placed at the head of this list and old items are dropped from the tail. In addition to the time sorted list, we maintain a hash index on the valid items that point to the set of their tags. Furthermore, for each tag, we keep a sorted index list of items that have been annotated with this tag. Let l_i represent the list maintained for tag t_i. l_i is sorted based on $score(d, t_i)$ for each item d, in descending order. When an item expires, it is also removed from the sorted list it belongs to. Considering newly arriving items is easily achievable as it causes only insertions to a few lists plus one insertion to the hash index and the time sorted list, as described above. Note that as opposed to standard top-k processing where each document has potentially very many features (terms), here, the average number of tags per item is rather small. As a result updating the structures with new arrivals does not incur high cost.

For the query execution we employ the threshold algorithm (TA) [12], which works as follows. It reads in parallel from the index lists, which are sorted by score in descending order. For each item observed it looks up its score in all other lists it has not been observed so far, which is done in our case with one lookup to the hash map as described in the previous paragraph. The aggregated scores of the items at the current sequential access scan depth define the stopping condition. The computation can be stopped if there are at least k items with a score better than the aggregated score at the sequential scan lines. We employ the TA algorithm over the single term index lists as our baseline algorithm.

The top-k query needs to be re-evaluated in two cases: first, when an item which was part of the top-k results expires. The second case happens when the set of top tags changes and causes a change in the query aggregation function. In order to avoid re-computations from scratch when a hot item expires, a k-skyband over the score-time space can be kept [28]. The k-skyband of a query contains only those items which have a chance of becoming a top-k result during their life time. When an item which was part of the top-k results expires, it is enough to evaluate the query on the k-skyband, instead of the entire valid items, to fill in the top-k results. This dramatically decreases the cost of re-evaluations, however, it is only useful when the query remains unchanged. For the rest of this paper we do not consider possible optimizations when the top-k query is not changing, as this is a well addressed problem [11, 28], rather, we will focus on solutions for the changing query issue. In the next section we describe our approach for pre-aggregating stable parts of the top-k query in order to decrease the cost of evaluations when the query changes.

4 Grouping for pre-aggregation

Pre-aggregating index lists for certain tag sets has the potential to greatly reduce the access cost at query processing time, as scores are already aggregated, less index lists have to be accessed, and the aggregated scores help to identify the final top-k result earlier, i.e., cause an earlier stopping of the algorithm. In the best case, one can simply cache the results of the entire query and re-use it many times. However, this is not very likely to happen, in our envisioned scenario. A good choice of which index lists to pre-aggregate should reflect the following two rationales.

1. Pre-aggregate only those parts that are very likely to occur in future queries, too. To this end, we monitor how consecutive queries overlap and identify tags which are stable to some extent: Observing the changes in the top-k query itself, which is considered to be quite large ($\sim$100 tags), shows that although the query itself changes more or less every time, there is a fraction of tags that remain as part of the query for a long duration of time. These consists of those tags which are popular most of the time and represent current long-lived events. Observing stable sub-queries, motivates us to maintain pre-aggregations for those sub-queries which can later be used to evaluate the complete query more efficiently.
2. While the previous step identifies candidate tags for pre-aggregation, not all of these tags should be used immediately, and, even more importantly, it remains unclear how these tags should be grouped together to find suitable pre-aggregations. As we will further discuss in this section, grouping tags together whose index lists show a certain degree of overlap, in terms of documents contained, is likely to be more beneficial than grouping tags whose index lists are mainly disjoint, as in the latter case, the aggregation boils down to a simple merge with no score aggregation performed.

In this section we propose to group lists corresponding to "stable" tags together to reuse their aggregated results. More precisely, we pre-aggregate certain lists and try to assemble at query time the final top-k result given the pre-aggregated values. At the point the evaluation is triggered, the query will be updated based on the new hot tags and then the hottest documents will be retrieved. While in general the query can change with each and every new document in the system (or an old document expiring), we issue an evaluation process only after a certain amount of new documents, e.g., 500 in our experimental evaluation. This is of course an application dependent parameter and could be chosen based on the capabilities of the computing server, for instance.

4.1 Optimal solution

To better understand the complexity of the problem, in this section, we formulate an offline algorithm that assumes complete knowledge of data arriving in the system in the future. With such knowledge, an optimal choice of pre-aggregations is possible, as the set of different top-k queries for a given time period is known to the algorithm. This is obviously an impossible assumption, but nevertheless discussed here to underpin the complexity of the problem and to better understand the approach considered in this paper.

Given a set of queries $\mathcal{Q} = \{Q_1, Q_2, \ldots, Q_n\}$, which correspond to the hot item (tag) sets for different time points, the goal is to find an optimal set of subsets of tags $\mathcal{S}$ that can answer all queries in Q efficiently, re-using pre-aggregations in $\mathcal{S}$. Each member of $\mathcal{S}$ is a subset of tags and if its cardinality is larger than one, represents a pre-aggregation of the lists maintained for the tags it contains. For example $\mathcal{S} \ni S_i = \{t_j, t_k\}$ means we are maintaining a sorted list for $t_j \vee t_k$. Let L_i represent the list corresponding to S_i. Items in L_i are sorted based on their score with regard to S_i: $s(d, S_i) := \sum_{t \in S_i \cap T_d} score(d, t)$.

In case of ties, more recent items are preferred. L_i is created utilizing the simple lists we maintain for tags which are members of S_i. Assuming equal length l for all simple lists, the cost of aggregating k such lists is $k * l$.

Now assume a query Q_y. Recall that each query is specified by a set of tags. We say a subset $S'_y \subset \mathcal{S}$ *exactly covers* Q_y if members of S'_y are pairwise disjoint and $\bigcup_{S_i \in S'_y} S_i = Q_y$. If a subset exactly covers a query, a standard TA algorithm can use it to evaluate that query. The effectiveness of a list L_i depends on the co-occurrences of tags in S_i in the stream of items. We assume the percentage of items likely to be read before TA can stop is known for a list L_i and we denote it by c_i. Note that c_i depends on the query and other available lists, but for simplicity we consider it as an independent fixed value. The cost of evaluating a query Q_y using S'_y can be estimated by: $\sum_{S_i \in S'_y} c_i$.

Let $\mathcal{P}$ be the powerset of $\bigcup Q_i$. Given the above cost functions, we can formulate our goal as an optimization problem which aims at minimizing the following cost function with regard to the boolean variables x_{ij}:

$$\sum_{S_i \in \mathcal{P}} y_i * |S_i| * l + \sum_{Q_j \in \mathcal{Q}} x_{ij} * c_i$$

and the following constrains:

$$\begin{cases} y_i = \bigvee_i x_{ij} & \text{(C1)} \\ \forall Q_j \ \forall t \in Q_j \ \sum_{i: t \in S_i} x_{ij} = 1 & \text{(C2)} \end{cases}$$

$x_{ij} = 1$ shows that S_i is used in evaluating Q_j. $y_i = 1$ if S_i is used in evaluating at least one query. The first constraint (C1) assures this. The first summation in the cost function accounts for the pre-aggregation expenses while the second part shows the evaluation cost. The second constraint (C2) ensures that the set of S_i's used for evaluating each query exactly cover that query.

The above optimization problem is not a standard linear programming problem, as the variables y_i depend on x_{ij}'s. However, even if we ignore the first part of the cost function (the query evaluation cost), we face a 0–1 linear programming problem which is known to be NP-hard (cf., e.g., [26]).

4.2 Efficient grouping

Given the complexity of the problem described above and the fact that the set of future top-k queries is actually not known in advance, we address the problem with an approximate approach.

Clearly it is beneficial to pre-aggregate sets of tags which frequently appear in the future top-k queries: Aggregating the corresponding lists of a set of tags pays off only when the resultant list can be used enough number of times in future queries. For each observed tag we maintain the number of times it has appeared in the set of hot tags and predict its probability of being part of the aggregation query based on this past information.

Assume the number of single tags with probability of appearing in future queries larger than a specific threshold is r. These tags have to be grouped together to form a pre-aggregated list. However, grouping all of them together may not be beneficial, as to be able to use such a pre-aggregation *all* involved tags should be part of the query. The probability that a pre-aggregation of m single lists is usable in future queries, decreases with increasing m: if p is the probability of the most frequent tag, and assuming tags appear independent of one another, p^m is an upper bound of the probability that this aggregation list is usable. We should therefore, pre-aggregate *subsets* of the r candidate tags.

Grouping those tags which co-occur together in the streaming items is highly beneficial for the overall performance as they have higher chances of appearing *together* in future queries. Given the data-driven nature of the query, the query evaluation using the TA algorithm can be done more efficiently due to the already pre-aggregated partial queries. A pre-aggregation of tags which do not co-occur together and aggregating them creates a list of size of sum of the sizes of single lists with non-aggregated scores. On the other hand, aggregating single lists which have high correlation, i.e., their corresponding tags occur together, results in a list with more score variations (in case of ties in the original list) and higher scores, which is more effective in decreasing the threshold value maintained by the TA algorithm and causing it to stop reading more entries.

As a measure of tag co-occurrence we calculate the resemblance value for two index lists, which is defined as the fraction of the size of their intersection over the size of their union. Based on the given intuitions above, in the next section we describe our proposed algorithm for selecting tag sets to be materialized.

4.3 Tag set generation

To actually compute the tag sets to be materialized, our algorithm considers all tags that occur in the queries with a probability above a parameter α. Since the cardinality of the set

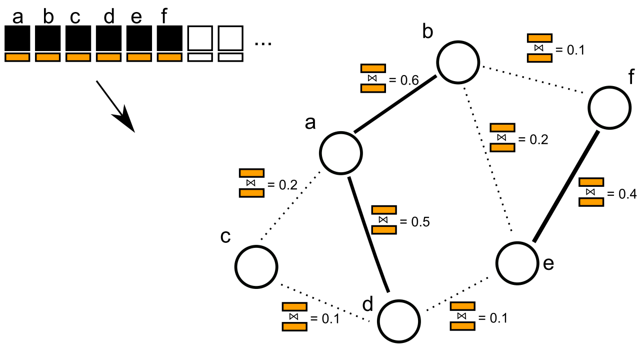

Fig. 2 Illustration (showing a subset of all cases to be considered) of the process of determining tag sets of interest for pre-aggregation for $\rho = 0.3$. Given the top re-occurring tags as the nodes in a graph, we connect those nodes whose index lists have a resemblance of at least 0.3. All resulting connected components are then selected and the corresponding index lists pre-aggregated

of tags is not large, we can maintain exact statistics for the number of occurrences of each tag in a query. We normalize the number of occurrences and use it as the probability of a tag's occurrence in future queries. As described above, these tags are considered candidates for pre-aggregation.

To eventually decide which groups of candidate tags to build and then to pre-aggregate depends on the amount of correlation (i.e., essentially overlap) of the particular index lists. Only those tags should be pre-aggregated together that show a sufficient pairwise correlation. Figure 2 illustrates our approach which consists of the following four steps.

1. each tag is considered to be a node of a graph
2. for each pair of tags the resemblance is calculated
3. each pair of tags with resemblance $\geq \rho$ is treated as an edge in a graph
4. the connected components of the graph are sets of tags to be materialized

This technique favors those frequently reoccurring parts of the query that also frequently appear *together* in the data stream.

4.4 FM sketches for resemblance calculation

As the computation of the exact resemblance is extremely expensive we employ a sketching technique that can efficiently estimate the resemblance value independent of the size of the involved index lists. In addition, as even exact resemblance numbers cannot guarantee the optimal pre-aggregation, the effect of slightly inaccurate resemblance numbers are negligible.

We make use of the well known Flajolet-Martin sketches (FM sketches) [13], which are compact and precise estimators of the cardinality of a multi-set. Given two sets S_1 and S_2 and their corresponding synopses in form of FM sketches, once can determine the size of the intersection by combining the sketches in an extremely efficient bit-wise fashion. More precisely, one obtains actually the size of the union given the bit-wise OR operation of the bit-sets of the two sketches. Then, the size of the intersection is given by the inclusion-exclusion principle ($|S_1 \cup S_2| = |S_1| + |S_2| - |S_1 \cap S_2|$), hence we can estimate the resemblance value.

As we keep index lists for the tags we observe, there is only the small overhead of maintaining a sketch for each of these lists. When enumerating the candidate tag sets we estimate their suitability to the query processing based solely on the sketches. There is no need to compute the aggregation and assess its size as the size is directly given by the sketch combinations, which is very efficient.

Due to the inherently approximate nature of the sketches, the resemblance values are not exact, which leads to decisions of which tag sets to materialize that varies from the algorithm employing the true resemblance numbers.

5 Personalizing hot-*k* queries

While the presented technique so far allows for an efficient retrieval of the best fitting documents to a continuously adapted top-*k* query derived from the tag usage frequencies, it does not allow for any form of personalization. The retrieved documents reflect merely a general (global) importance. However, with the wide spread interest in Web 2.0 technologies spreading across different domains and touching various kinds of user interests, some kind of personalized information delivery is needed to ensure usability.

Given the presented approach, we are able to compute the essence of current information, in form of document lists reflecting the currently hot topics. We consider this as the underlying mechanism to automatically generate a Website to give users a glimpse of currently interesting articles. This is in contrast to pure information filtering, which matches user provided subscription queries to an incoming stream of documents [15, 27, 30] and delivers only the best top-*k* matching documents to a user.

Existing approaches on personalized information filtering assume users register with keyword queries, or the so called profiles or subscriptions, in the system. The straightforward way to implement this it to let profiles, of the form $p = (u_1, \ldots, u_m)$, specify a normalized weight u_i for a keyword k_i, where k_i is one possible keyword in the universe of keywords $\mathcal{K}$. For simplicity we assume $\mathcal{K} = \mathcal{T}$, and will use the terms "tags" and "keywords" interchangeably. To return the top-*k* results for each registered query, each incoming document is evaluated against all stored profiles. Assume d is a new incoming document. If d's score with regard to a profile p is larger than p's ranked k result, d is inserted into p's result set. Furthermore, each time an item which was part of the result set of a profile expires, that profile should be re-evaluated against the set of valid items.

The methods in [27] and [15] are extensions to a brute-force evaluation which avoid evaluating each incoming document against *all* profiles by organizing the profiles in a way that reflects the current quality of the top-*k* results. Thus, these methods can assess upfront, if a document is useful for certain profiles or can be dropped without further consideration in order to avoid unnecessary evaluation costs. In this section, we discuss how the profile filtering approach presented in our previous work [15], can be adapted to provide efficient re-ranking of the hot-*k* items, and consequently personalize the set of hot-*k* items.

Let S_h denote the stream of hot items which are the result of the previously defined dynamic hot-*k* query. The goal is to re-rank these documents based on user preferences, formulated as profiles as explained above. Each item in S_h has an associated score $s(d, H_\tau)$ as defined by (1) of Sect. 1. This score is independent of user profiles. In order to account for user interests, for each hot item, we define the following *profile score* which reflects the relatedness of a hot document d to a given profile p:

$$s(d,p) := \sum_{t_i \in \mathcal{T}_d} score(d,t_i) * u_i \qquad (2)$$

To personalize the stream of hot items for each user, we aim at re-ordering them for each profile, based on the above profile related score, in addition to the hotness score. We call this *personalized hotness score* and represent it with $s_h(d,p) = s(d,p) + s(d,H_\tau)$.

User profiles in reality contain very few non-zero weighted terms (around 3–5). We use this property to have an immediate speed up on query evaluations by considering each document against only those profiles which share a non-zero weighted term with that document. To implement this, we maintain an inverted profile index: for each tag t_i, we keep a list k_i of profiles in which t_i has a non-zero weight. Now, to calculate $s(d,p)$, we only need to process profiles in lists corresponding to non-zero weighted tags in item d. For other profiles $s(d,p) = 0$, and $s_h(d,p) = s(d,H_\tau)$ which is the score associated with the hot item d as it arrives.

We avoid some unnecessary computations by utilizing the traditional inverted indices as explained above. However, we can gain further efficiency by ordering the profiles in each list in such a way that allows us to avoid processing all profiles in a list. The profile indexing method introduced in [15] is built around this idea and uses an "early stop" algorithm to prevent unnecessary evaluations. We adapt this approach for our problem to further improve efficiency.

First, note that we can write $s_h(d,p)$ as $\sum_{t_i \in \mathcal{T}} score(d,t_i) * (u_i + I(t_i))$, where I is an indicator function,

$$I(t_i) = \begin{cases} 1 & t_i \in H_\tau \\ 0 & otherwise \end{cases} \qquad (3)$$

which takes the value 1 if t_i is a hot tag and 0 otherwise.

Let $p.s$ denote the personalized hotness score of p's ranked k item. In the list corresponding to tag t_i, we keep the profiles sorted based on the value $(u_i + I(t_i))/p.s$ where u_i is t_i's weight in profile p and I is the indicator function defined above. Now, assume a new hot item d with set of tags $\mathcal{T}_d$ arrives in the system. We do sorted accesses in a round robin fashion to all inverted lists k_i where $t_i \in \mathcal{T}_d$. When a profile p is seen under one of these lists, we calculate $score(d, p)$ and update $s_h(d, p)$. If $s_h(d, p)$ is larger than the personalized hotness score of the ranked k item in p's result set, we insert d in the ordered result set of profile p based on $s_h(d, p)$. While accessing the sorted lists in a round robin fashion, we also check the following stopping condition. For a list k_i let $\underline{v_i}$ be the last observed value under sorted access. We stop the above procedure when $score(d, t_1) * (\underline{v_1}) + \cdots + score(d, t_m) * (\underline{v_m}) < 1$.

As our personalized hotness score is monotonic, we can use the result of [15] to show that with the above stopping condition, for each profile, we can still return the exact top-k results. Each time the set of hot tags changes, the profile lists corresponding to the new hot tags and those corresponding to tags which are not anymore hot, are updated. This is necessary, as the value of the indication function for these tags changes. However, as previously discussed, we expect the changes to set of hot tags to be limited, i.e., affecting only few tags and their lists. Therefore, the amortized cost of updating the profile lists over time is not large. With the above procedure, we are able to return to each user an ordered list of hottest documents, where ordering is based on both the hotness of the document, as defined by the dynamic query, as well as user interests which are represented as profiles.

6 Experiments

We have implemented our algorithm in Java 1.6 and executed on a Windows 2003 server with a quad core 2.33 GHz Intel Xeon CPU, 16 GB RAM, and a 800 GB RAID-5 disk.

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 etc. 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 11,395,571 (timeStamp, postId, tags)-entries, with 2,444,780 distinct postIds, hence, an average of ~ 2.2 tags per blog entry. This dataset does not contain duplicate tags per document. That means, the score of a document w.r.t. a particular tag is considered to be 1 if the tag is attached to the document, 0 otherwise. Ideally, the above dataset would reflect every single tag assigned from users to documents, hence, would contain in general multi-sets of attached tags. However, the tags here are more like topics/categories. Nevertheless, we opted for using this standard data set for reproducibility reasons. Note that our approach is not limited to operating over boolean values and can be used also in presence of multiple occurrences of a tag in a document, in which case we would consider also tag frequency measures, similar to term frequencies known from standard information retrieval approaches.

6.1 Algorithms

We consider the performance of three algorithms in this experimental evaluation. All are based on the TA algorithm [12]. The difference stems from the index lists they can involve in the query processing. More precisely, we run the following algorithms:

- *plain*: This is the plain algorithm involving only accesses to single-tag index lists.
- *comb*: This algorithm uses pre-aggregation of tag sets that are supposed to help the query execution. The set of tags to be pre-aggregated are chosen using the algorithm described in Sect. 4.3. True resemblance values are calculated by merging lists and measuring the resultant size.
- *combsketch*: This algorithm also uses pre-aggregation tag sets as described in Sect. 4.3. However, the resemblance values are estimated using sketches as described in Sect. 4.4.

Note that the comb algorithm is in fact impractical, as it incurs huge costs just for measuring the resemblance values. However we ignore this cost and use this algorithm to show the best achievable performance using our proposed set aggregation method.

6.2 Measures of interest

We will report on several measures as part of our performance study. Note that we do not report on accuracy measure as *all algorithms report the exact top-k results* to the query described above. We consider the number of entry accesses as the main cost to assess the suitability of the methods under comparison. We split this measure up in several

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

ingredients to better understand the strong and weak points of the approaches. In particular for the algorithms that use pre-aggregation, the cost for materializing lists for sets of tags does not occur in each query processing step. We measure:

- *eval_cost*: This measure reports on the average number of entry accesses the threshold algorithm makes to calculate the results.
- *pre-aggregation_cost*: With this measure we provide an insight on how costly the pre-aggregation operation is, that means, how many entries on average need to be accessed when materializing the index lists for sets of tags, determined by the selection algorithm. The plain algorithm does not incur any pre-aggregation cost.
- *total_cost*: In addition to the measures described above we also report on the total cost which consists of the total (non-averaged) cost for all query evaluations plus the overall cost for doing the pre-aggregation. We ignore the cost for calculating the resemblance values.

6.3 Results

We run the mentioned three algorithms for different parameter settings averaging over 45 query evaluations for each setting. The query evaluation is fired at every 500 items. The tag set generation algorithm (described in Sect. 4.3) is run periodically at every 20 evaluations. Unless otherwise stated we use a time-based sliding window of size $W = 10{,}000{,}000$ milliseconds. The default number of desired top-k items denoted by *kdocs* is 100. The number of tags used in defining the query is denoted by *ctags* and its default value is set to 75.

We first observe the effects that parameters α and ρ have on the costs incurred by our proposed algorithms. α represents the threshold value we use in generating the tag sets. As explained in Sect. 4.3, we only consider tags which have a higher probability of occurrence in queries than α for tag set generation. ρ, on the other hand, represents the threshold we use to mark a pair of tags as connected in the graph used for tag set generation. Figures 3, 4, and 5 shows the different cost values while varying the parameter α and fixing all other parameters. As explained in Sect. 4.3, α denotes the threshold for considering a tag for the subsequent tag set generations. Figure 3 presents the evaluation cost with changing α. Small α values causes the algorithm to consider tags which actually do not occur later in the query. These tags may have high enough resemblance with other tags to be part of a connected component. The tag set corresponding to such a component is however, useless, since it contains a tag which does not actually appear in the query. As a result, both comb and combsketch have total costs close to plain with small α values. On the contrary, for large enough values of α a large fraction of materialized lists are in fact reusable, therefore the evaluation cost of comb and combsketch is much smaller than plain. For too high values of

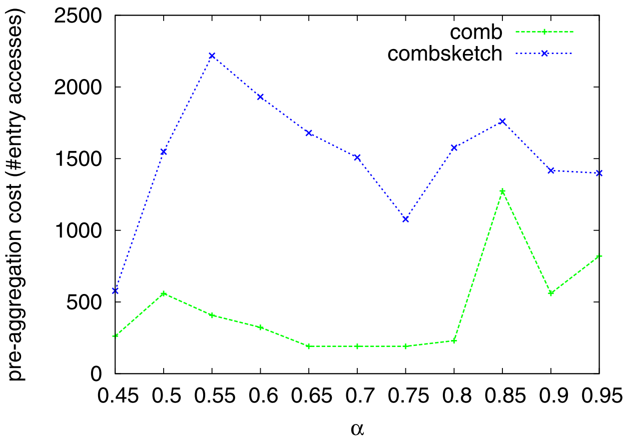

Fig. 4 Pre-aggregation_cost values when varying the α parameter. $W = 10{,}000{,}000$ ms, kdocs = 100, ctags = 75, $\rho = 0.6$

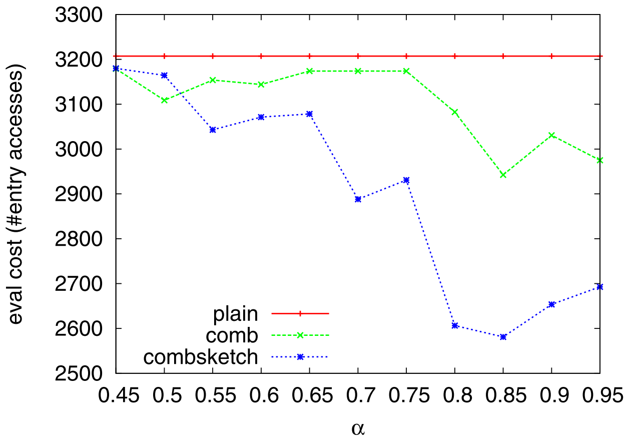

Fig. 3 Eval_cost values when varying the α parameter. $W = 10{,}000{,}000$ ms, kdocs = 100, ctags = 75, $\rho = 0.6$

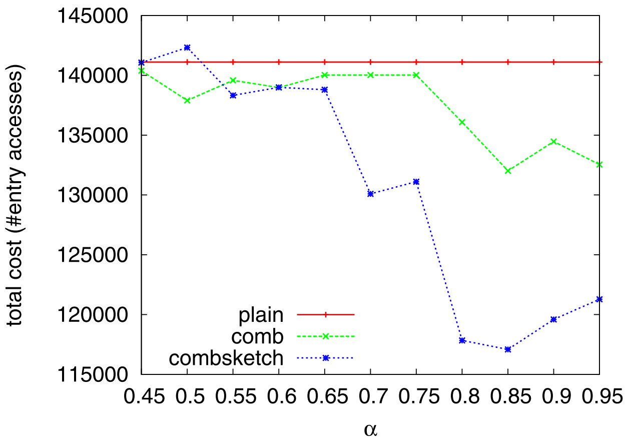

Fig. 5 Total_cost values when varying the α parameter. $W = 10{,}000{,}000$ ms, kdocs = 100, ctags = 75, $\rho = 0.6$

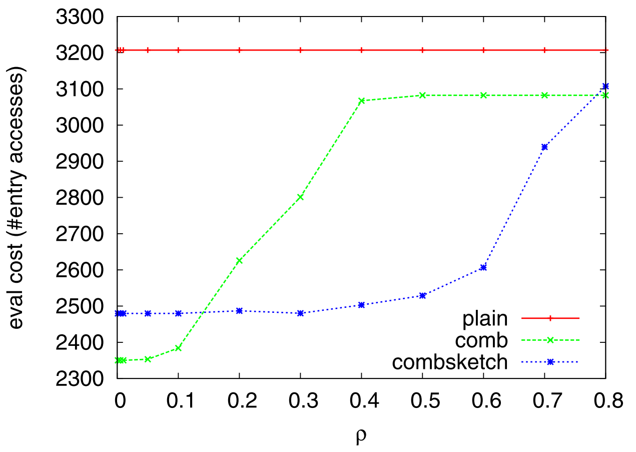

Fig. 6 Eval_cost values when varying the ρ parameter. $W = 10{,}000{,}000$ ms, kdocs = 100, ctags = 75, $\alpha = 0.8$

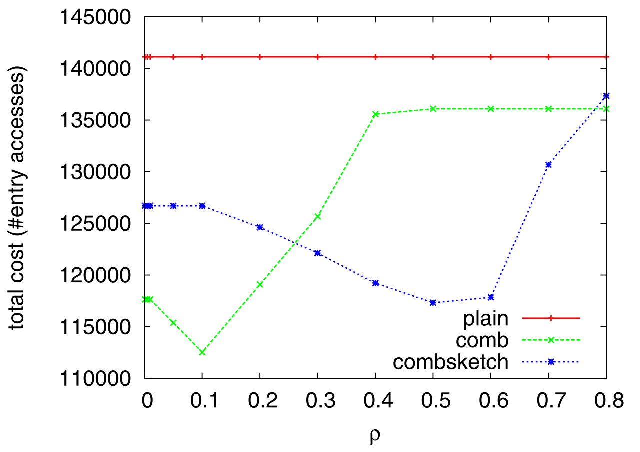

Fig. 8 Total_cost when varying the ρ parameter. $W = 10{,}000{,}000$ ms, kdocs = 100, ctags = 75, $\alpha = 0.8$

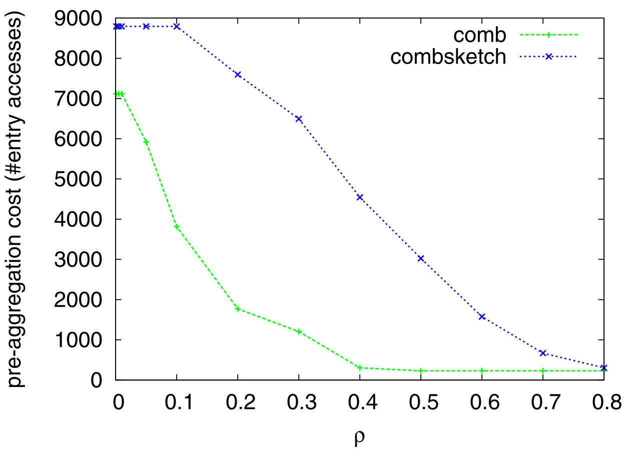

Fig. 7 Pre-aggregation_cost when varying the ρ parameter. $W = 10{,}000{,}000$ ms, kdocs = 100, ctags = 75, $\alpha = 0.8$

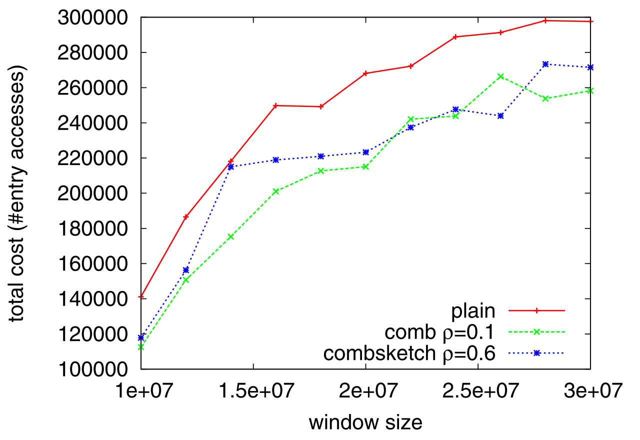

Fig. 9 Total_cost when varying the window size, $\alpha = 0.8$, kdocs = 100 and ctags = 75

α, less than necessary number of tags are actually considered, which results in lowering the total benefits of them in evaluating the queries. The pre-aggregation_cost is shown in Fig. 4. We see that for $\alpha = 0.85$ both comb and combsketch have high pre-aggregation_cost which actually pays off very well, as the total cost at this value has a minimum for both methods.

Figure 8 shows the total costs when varying the parameter ρ, which specifies whether or not an edge should be considered between two nodes in the tag set generation algorithm. In our experiments ρ is not an absolute value, as the resemblance values estimated by combsketch and sketch are very different in the absolute sense but they usually hold the same ordering: if a list l_1 has higher true resemblance to l_2 than l_3 this likely holds also in the estimated values by combsketch. So we calculate the highest resemblance value res_{max} and $\rho * res_{max}$ is the threshold considered. We repeat the same procedure for $\rho + step$, each time increasing the resemblance threshold until it reaches 1. This way, we produce smaller tag sets which have high resemblances. So, as observed also in Fig. 7 the pre-aggregation cost decreases by increasing ρ. Note that the ρ value were the pre-aggregation cost is actually paid off in evaluations is different for comb and combsketch.

In contrast to the behavior of the pre-aggregation cost, the eval_cost increases with larger values of ρ, as shown in Fig. 6. This is expected, as with larger ρ values less and less tags are put together for pre-aggregation and, hence, our methods converge to the plain method.

After discovering good parameters for our algorithms, we evaluate our methods by fixing those parameters to the best found, and changing the system variables. Figure 9 shows the total cost incurred by the three algorithms when changing the size of the sliding window. Clearly the cost for all three methods increases, as more items are valid at each instance of time, therefore the lists to be accessed are longer. However our algorithms incur much less cost than the plain algorithm. Figure 10 shows the same measure when chang-

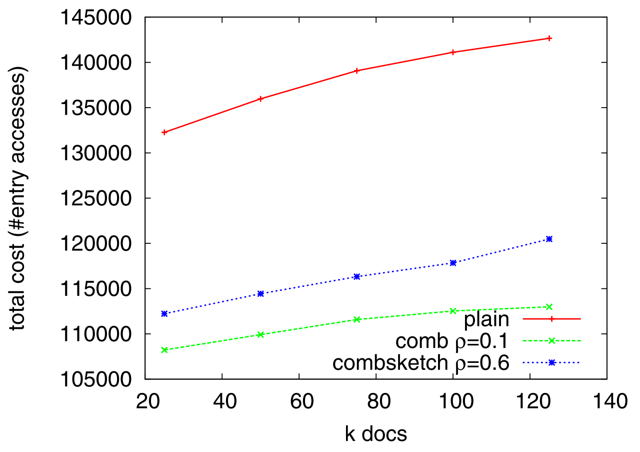

Fig. 10 Total_cost when varying kdocs, $\alpha = 0.8$, $W = 10{,}000{,}000$ ms and ctags = 75

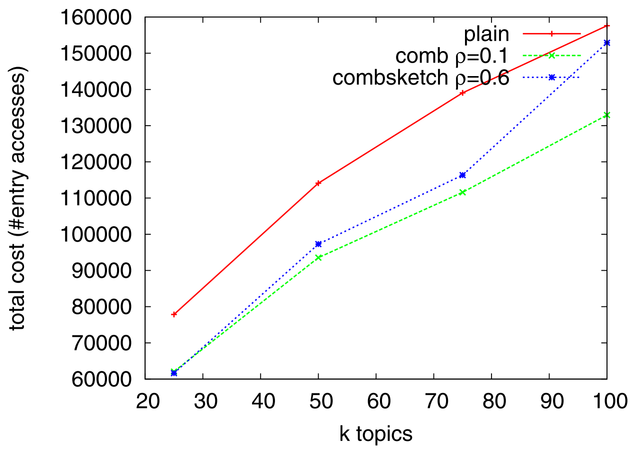

Fig. 11 Total_cost when varying ctags, $\alpha = 0.8$, $W = 10{,}000{,}000$ ms and kdocs = 75

ing kdocs. As expected the TA algorithm can stop earlier for smaller values of kdocs. Figure 11 finally, shows the total cost when varying ctags. Since this number defines the number lists we should consider in the evaluation, it has a direct effect on total cost. In all three cases, our proposed algorithms incur less cost than the plain method. Although combsketch has only estimates of the true resemblances, its performance gains is very close to comb which has the true resemblance values.

7 Conclusion and future work

With the immense rate at which new information are published nowadays, in particular pushed by the widespread use of blogs, social networks, and other messaging services, it becomes harder and harder to stay tuned to the essence of what is going on in the world. We addressed the problem of continuous monitoring of top-k hottest items over a stream of tagged items such as blog entries or images. We have defined the property of being hot as a top-k aggregation query where the query itself is characterized by the set of most popular tags in a given time period. This causes the top-k query to change over time, hence requires the system to re-evaluate the top-k query from scratch. Our approach is based on the observation that parts of the top-k query are stable for certain time intervals, therefore, do not have to be re-computed in each evaluation phase. As materializing pre-computations of all possible subsets is impractical, we have presented an approximate algorithm to identify the most promising tag subsets (i.e., top-k query ingredients) leveraging FM sketches to predict the suitability of these tag sets. The presented generation method itself gives an easy to use mean to control the amount of pre-aggregated lists. To enhance the hot document tracking, we have described a viable approach to personalized information filtering of these hot documents, enabling a re-ordering of hot documents based on user defined criteria.

In this paper, we treated the set of hot tags as one large query over the incoming streams. An alternative would be to consider several smaller queries, each corresponding to a set of co-occuring hot tags and returning the top-k documents according to an aggregate of their scores with regard to these smaller queries. Unlike the case with only one big query, just building these queries requires maintaining extra information which allows the system to extract the hot co-occuring tags. A possible subject for future work can consist of comparing these two approaches. On the other hand, from a semantics point of view, it would be interesting to investigate which of the above schemes is more successful in returning the interesting documents to the user.

References

1. Allan J, Carbonell J, Doddington G, Yamron J, Yang Y (1998a) Topic detection and tracking pilot study final report. Computer Science Department. Carnegie Mellon University. Paper 341. http://repository.cmu.edu/compsci/341
2. Allan J, Papka R, Lavrenko V (1998b) On-line new event detection and tracking. In: SIGIR, pp 37–45
3. Alon N, Gibbons PB, Matias Y, Szegedy M (2002) Tracking join and self-join sizes in limited storage. J Comput Syst Sci 64(3):719–747
4. Alvanaki F, Michel S, Ramamritham K, Weikum G (2011) Enblogue—emergent topic detection in Web 2.0 streams. In: SIGMOD conference
5. Babcock B, Babu S, Datar M, Motwani R, Widom J (2002) Models and issues in data stream systems. In: PODS, pp 1–16
6. Börzsönyi S, Kossmann D, Stocker K (2001) The skyline operator. In: ICDE, pp 421–430
7. Calders T, Dexters N, Goethals B (2007) Mining frequent itemsets in a stream. In: ICDM, pp 83–92
8. Charikar M, Chen K, Farach-Colton M (2004) Finding frequent items in data streams. Theor Comput Sci 312(1):3–15
9. Cormode G, Muthukrishnan S (2003) What's hot and what's not: tracking most frequent items dynamically. In: PODS, pp 296–306

10. Das G, Gunopulos D, Koudas N, Tsirogiannis D (2006) Answering top-*k* queries using views. In: VLDB, pp 451–462
11. Das G, Gunopulos D, Koudas N, Sarkas N (2007) Ad-hoc top-*k* query answering for data streams. In: VLDB, pp 183–194
12. Fagin R (2002) Combining fuzzy information: an overview. SIGMOD Rec 31(2):109–118
13. Flajolet P, Martin GN (1985) Probabilistic counting algorithms for data base applications. J Comput Syst Sci 31(2):182–209
14. Flickr, photo sharing: http://www.flickr.com
15. Haghani P, Michel S, Aberer K (2010) The gist of everything new: personalized top-*k* processing over Web 2.0 streams. In: CIKM, pp 489–498
16. Haghani P, Michel S, Aberer K (2011) Tracking hot-*k* items over Web 2.0 streams. In: BTW, pp 105–122
17. He Q, Chang K, Lim EP (2007) Analyzing feature trajectories for event detection. In: SIGIR, pp 207–214
18. Hotho A, Jäschke R, Schmitz C, Stumme G (2006) Trend detection in folksonomies. In: SAMT, pp 56–70
19. Hristidis V, Koudas N, Papakonstantinou Y (2001) Prefer: a system for the efficient execution of multi-parametric ranked queries. In: SIGMOD conference, pp 259–270
20. Jin C, Yi K, Yu JX, Lin X (2008) Sliding-window top-*k* queries on uncertain streams. PVLDB 1(1):301–312
21. Kleinberg J (2006) Temporal dynamics of on-line information streams. In: Data stream management: processing high-speed data. Springer, Berlin
22. Kumar R, Novak J, Raghavan P, Tomkins A (2005) On the bursty evolution of blogspace. World Wide Web 8(2):159–178
23. Kumar R, Punera K, Suel T, Vassilvitskii S (2009) Top-*k* aggregation using intersections of ranked inputs. In: WSDM, pp 222–231
24. Mathioudakis M, Koudas N (2009) Efficient identification of starters and followers in social media. In: EDBT, pp 708–719
25. Mathioudakis M, Koudas N (2010) Twittermonitor: trend detection over the twitter stream. In: SIGMOD conference, pp 1155–1158
26. Mehlhorn K, Sanders P (2008) Algorithms and data structures: the basic toolbox. Springer, Berlin
27. Mouratidis K, Pang H (2009) An incremental threshold method for continuous text search queries. In: ICDE, pp 1187–1190
28. Mouratidis K, Bakiras S, Papadias D (2006) Continuous monitoring of top-*k* queries over sliding windows. In: SIGMOD conference, pp 635–646
29. Muthukrishnan S (2005) Data streams: algorithms and applications. In: Foundations and trends in theoretical computer science. Now Publishers Inc
30. Yan TW, Garcia-Molina H (1994) Index structures for selective dissemination of information under the boolean model. ACM Trans Database Syst 19(2):332–364
31. Yi K, Yu H, Yang J, Xia G, Chen Y (2003) Efficient maintenance of materialized top-*k* views. In: ICDE, pp 189–200
32. Youtube, broadcast yourself: http://www.youtube.com/

Parisa Haghani received her Ph.D. degree in Computer Science from Ecole Polytechnique Fédérale de Lausanne (EPFL, Switzerland) in August 2010. Her research interests include efficient processing of ranking queries in novel applications, more specifically in data streams and peer-to-peer networks. Parisa received a B.Sc. in Computer Engineering from Isfahan University of Technology (IUT, Iran) in August 2004 and an M.Sc. in Artificial Intelligence in from Sharif University of Technology (SUT, Iran) in August 2006.

Sebastian Michel is an independent research group leader at Saarland University, Saarbrücken, Germany. He received his diploma in computer science from the University of Marburg, Germany, in 2004. From 2004 on he was a doctoral student at the Max-Planck-Institute in Saarbrücken, where he received his Ph.D. in 2007, summa cum laude. For his Ph.D. thesis, Sebastian received two awards; the Otto-Hahn medal of the Max-Planck Society (MPG) and the GI DBIS Dissertation award. He joined EPFL Lausanne, Switzerland, in 2007 as a post doctoral researcher before joining Saarland University, Germany, in 2009, building up his own research group. Sebastian's research directions include data mining in streams on the Web, distributed information systems such as peer-to-peer networks, sensor networks, and multimedia information retrieval.

Karl Aberer is a full professor for Distributed Information Systems at EPFL Lausanne, Switzerland, since 2000. Since 2005 he is the director of the Swiss National Research Center for Mobile Information and Communication Systems (NCCR-MICS, www.mics.ch). Prior to his current position, he was senior researcher at the Integrated Publication and Information Systems institute (IPSI) of GMD in Germany. He received his Ph.D. in mathematics in 1991 from the ETH Zürich. His research interests are on semantics and self-organization in information systems with applications in peer-to-peer search, semantic web, trust management and mobile and sensor networks.