# Filtering Structures for Microblogging Content

Ryadh Dahimene and Cédric du Mouza

CEDRIC Laboratory – CNAM

firstname.lastname@cnam.fr

2 rue Conté, F75141, Paris, France

Abstract

In the last years, microblogging systems have encountered a large success. After 7 years of existence Twitter claims more than 271 million active accounts leading to 500 million *tweet*s per day. Microblogging systems rely on the *all-or-nothing* paradigm: a user receives all the posts from an account s/he follows. A consequence for a user is the risk of *flooding*, *i.e.*, the number of posts received from all the accounts s/he follows implies a time-consuming scan of her/his feed to find relevant updates that match his interests. To avoid user flooding and to significantly diminish the number of posts to be delivered by the system, we propose to introduce filtering on top of microblogging systems. Driven by the shape of the data, we designed different filtering structures and compared them analytically as well as experimentally on a *Twitter* dataset which consists of more than 2.1 million users, 15.7 million tweets and 148.5 million publisher-follower relationships.

*Keywords:* Microblogging, Filtering, Indexing, Scalability issue

# Filtering Structures for Microblogging Content

Microblogging systems have become a major trend as well as an important communication vector. In less than seven years, *Twitter*[1] has grown in a spectacular manner to reach more than 500 million users in August, 2013[2] from which 54% are active users[3] (posting at least a *tweet* per month). Other similar systems like *Sina Weibo* [4], *Identi.ca*[5] or *Plurk*[6], to quote the largest, also exhibit dramatic growth. In such systems, the length of a published piece of news (called by post in the following) is limited to 140 characters which corresponds on average to 14.7 terms (Foster et al., 2011), so clearly greater than 4-5 terms adverts as reported in (König, Church, & Markov, 2009) but smaller than RSS items (Hmedeh et al., 2011), blogs (Ma & Zhang, 2007) or Web pages which have a size of 450 to 500 terms excluding tags (Levering & Cutler, 2006). The rationale for using such small messages is that these services were designed to be accessed with traditional cell phones through Short-Text Messaging protocol[7].

In microblogging, a user, represented by his account, follows other accounts to be notified whenever they publish some information. Conversely, s/he becomes a publisher for the accounts that follow her/him, which results in the existence of a large social graph. One of the main differences with other online social networks is that in microblogging, the graph linking the users can be seen as an interest graph, since users follow each other if they have interest in each other posts. Those networks work on an *all-or-nothing* fashion, i.e. if a user A follows a user B, A will receive all of B's posts.

Microblogging is also characterized by the heterogeneous nature of users. In *Twitter* for instance, there exist some high update frequency accounts (newspapers, tech-blogs and journalists) and others that publish less than one post a week. Moreover, there exist very popular accounts (*e.g.* news channel accounts like @FoxNews[8] or famous accounts like @BarackObama[9] with more than 45 million followers), and others with 0 or 1 follower.

For various reasons (security, advertisement, control policy …) these systems rely on a centralized architecture[10]. Each post published is received by the central system that forwards it to *all* the followers of the publishing account[11]. The central system is also responsible of the search feature (Busch et al., 2012). Since the most active accounts are generally the ones with the highest number of followers, the system must face a tremendous amount of posts to forward. For instance Twitter, which claimed in 2011 more than 200 million tweets a day, had to deliver daily over 350 billion *tweets*[12]. This traffic overload (especially for high-followed accounts) represents from an architectural point of view a scalability bottleneck.

On the follower's point of view, the amount of posts received from the accounts s/he follows, between 30 and 200 depending on the system considered (Kwak, Lee, Park, & Moon, 2010) loses the reader in the middle of long feeds of posts. This results in poor data readability and potentially loss of valuable information.
In a UK-based questionnaire over 587 participant, Bontcheva, Gorrell, & Wessels (2013) report that 33.9% of users perceive that they receive too many posts and 70.4% have found the task of locating the interesting/relevant posts amongst the others difficult. Also, 66.3% of the interviewed users have felt at some time that they can't keep up with the amount of received tweets. A direct consequence of this phenomenon is the high dynamicity of the graph: to avoid

flooding, users who follow active accounts tend to unsubscribe because they can't manage the continuous flow of posts as studied in (Kwak, Chun, & Moon, 2011; Kivran-Swaine, Govindan, & Naaman, 2011).

In order to improve the user experience and reduce the network load, we have chosen to introduce filtering in microblogging systems. The main underlying idea is that a user *A* follows another user *B* for some topics, and consequently s/he wants to receive only a subset of *B*'s posts that matches his interest. Such a structure must efficiently retrieve for an incoming post all followers of a publishing account whose filter is satisfied by the post. While designing the filtering structures, we took a particular consideration about some specific aspects of microblogging systems which we can summarize as:

- **short messages:** the size restriction (generally 140 characters) means that we handle short documents, the average length of a *tweet* is 14.7 terms (Foster et al., 2011);
- **account heterogeneity:** microblogging studies have revealed account heterogeneity in term of both update frequency and number of followers;
- **graph evolution:** As observed in Twitter (Kwak et al. 2011; Kivran-Swaine et al. 2011), users follow and unsubscribe often to other accounts. The filtering structure must consequently handle the graph dynamicity in order to handle this phenomenon;
- **centralized system:** The social graph is stored by the microblogging system. This system receives all posts and forwards them to followers according to the graph it stores. That means the whole task is supported by the centralized system. Therefore we must reduce the filtering process time by trying to manage the matching in central memory.

Considering the main characteristics of such a system. The main contributions of our paper are:

i. the presentation of four indexes, based on traditional inverted lists approaches, which support content filtering in microblogging systems;
ii. analytical comparisons of the structures efficiency in term of memory requirements, matching and update times of the three structures;
iii. an experimental validation of the analytical costs for the structures on a dataset of 2.1 million users, 15.7 million tweets and 148.5 million following relationships.

The rest of the paper presents the detailed Related Work followed by the Data model where we define our system, including the definition of the filters as well as the notification process. Section indexing describes our filtering structures and their analytical study. Section Experiments presents our experimental setting and results followed the conclusion.

## Related Work

### *Microblogging systems characterization*

The impressive growth of microblogging systems with their tremendous amount of user generated content has attracted the research community. Java, Song, Finin, & Tseng (2007) presented one of the first studies that looked inside *Twitter* and showed for instance that users

with similar intentions tend to connect with each other. Kwak et al. (2010) have also studied the following behavior and information diffusion patterns for a large snapshot of the entire *Twittersphere*. We relied on those studies in order to get insights about the user behavior on microblogging systems.

*Search and indexing*

Some recent works tackle the scalability issue when searching relevant posts by indexing the flow of *tweets*, like *EarlyBird* (Busch et al., 2012) which describes the real-time indexing structure used by *Twitter* to implement the search function. It relies on an inverted index mapping keyword queries to append-only posting lists which contains tweets in the ascending order of their timestamps. This allows to retrieve effectively the newest tweets for a given query.

In *Tweet Index* (Chen, Li, Ooi, & Wu, 2011), the authors propose to reduce the indexing time without decreasing the quality of the search result by indexing immediately tweets which have a high probability to belong to a query, and by delaying indexing of other tweets. *Tweet Index* performs a classification task to rank tweets against the most popular previously encountered queries. This classification enables *TI* to select tweets to be indexed first by picking the tweets which matched the most popular queries.

In *LSII*, Wu, Lin, Xiao, & Xu (2013) adopt a similar approach. It implements a growing size index cascade in order to insert tweets from the smaller index and later move tweets in batch mode. Oppositely to these approaches which aim at performing (partial) matching between a query composed by several terms, and a tweet, we consider single-term filtering which allows different factorizations of inverted index.

*Ranking and recommendation*

Avoiding user flooding can also be achieved by providing the relevant content through recommender systems. Sriram, Fuhry, Demir, Ferhatosmanoglu, & Demirbas (2010) have presented a classification of tweets based on user profiles. It relies on a trained classifier to route an upcoming *tweet* to a predefined class. Sankaranarayanan, Samet, Teitler, Lieberman, & Sperling (2009) describe a technique for concept extraction from noisy sources, and apply it to retrieve news from *Twitter* data. In (Bakshy, Hofman, Mason, & Watts, 2011; Weng, Lim, Jiang, & He, 2010; Zhang, Li, & Wang, 2013) the authors propose metrics to compute the user influence in *Twitter* and to rank tweets according to the user influence. Uysal & Croft (2011) present a filtering approach which takes advantage of the retweet behavior to bring important tweets forward. Liang, Xu, Tjondronegoro, & Christen (2012) proposes to provide topic recommendations by leveraging microblogs implicit links (relationships between users, topics and posts) as well as temporal information from the posts. Vosecky, Leung, & Ng (2012) also used the implicit microblogs information combined with content-based features in order to filter and rank posts according to their quality. Another approach consists of using domain knowledge to provide recommendations (Bouraga et al., 2014), This approach presents the advantage of avoiding the "cold-start" problem (*i.e.*, not having enough historical data to infer recommendations). Finally, efficient community detection in social networks can be used to compute clusters in the users set and perform recommendations (Yin, Li, & Niu, 2014).

All these approaches aim at providing more quality content to users but do not diminish the number of delivered messages by the centralized system. Moreover, they require either large space or time overheads. Oppositely our approach proposes a structure that handles the filtering process on the server's side reducing drastically the number of posts delivered at cost of small overheads.

*Publish/subscribe systems*

In on-line publish/subscribe systems, different index structures have been proposed to handle the matching of incoming subscriptions. In *Le Subscribe,* Pereira et al. (2000) present a *Count-based* index which matches subscriptions by counting predicates that are satisfied. Broder et al. (2011) have introduced the evaluation of graph constraints in content-based publish/subscribe systems. The authors suppose that the publishers and subscribers are connected by a directed graph (like in microblogging systems) and they implement algorithms to efficiently evaluate constraints.

Silberstein, Terrace, Cooper, & Ramakrishnan (2010) consider high frequency update feeds. The authors propose a method to selectively materialize user events over streams in order to improve the scalability of matching algorithms in such systems. Another filtering technique is presented by Haghani, Michel, & Aberer (2010) where the authors process top-k algorithms on top of Web 2.0 streams which allow to filter out posts based on the relevance and the freshness within a sliding window. (Hmedeh et al., 2012) proposes and compares indexing schemes for a pub/sub system that scales to high publication rates. Our context is quite different since we have shorter messages (posts) , short queries (single-term filters) and a number of users and queries much more important in heterogeneous social graph. Additionally, in publish/subscribe the producers and consumers of data are distinct actors while in social networks a single user can play both roles.

## Data model

In this Section we introduce our micro-blogging model and our content filtering based on single-keyword queries.
Table 1 summaries notations that we will use through the rest of the paper.

*Microblogging system*

A micro-blogging system like *Twitter* can be represented as a directed graph $G = (N, E)$ where the set of nodes $N$ represents the users (accounts) of the system and the set of edges $E \subseteq N \times N$ represents the *following* relationships. More precisely a directed edge $e=(u, v)$ exists from a node $u$ to a node $v$ if the user whose account is $u$ is notified whenever the user whose account is $v$ publishes a piece of news ($u$ receives $v$'s updates).

| *Notation* | *Description* |
|---|---|
| $N$ | total number of accounts |
| $\Gamma^+(n)$ | accounts followed by n |
| $\Gamma^-(n)$ | accounts that follow n |
| $\varphi$ | average number of followers for a user |
| $\tau$ | average number of filter terms for a (publisher, follower) pair |
| $\|p\|$ | size (distinct terms) of a post p |
| $(k, \beta)$ | Heaps' law coefficients for micro-blogging datasets |
| $\gamma$ | Zipf's law constant for micro-blogging datasets |
| $V_F$, $V_P$ | filter vocabulary, posts vocabulary |
| $\theta_{dir}$ | size of a directory entry |
| $\theta_{list}$ | size of a posting list |
| $\theta_{entry}$ | size of an entry in a posting list |

Table 1. Notations

In the following we blur the distinction between a user, an account and a node. For a node $n$, we define $\Gamma^+(n)$, the set of nodes followed by $n$, *i.e.*, its successors in $G$ as

$$\Gamma^+ : N \rightarrow 2^N, \Gamma^+(n) = \{n' \mid (n, n') \in E\}$$

We define similarly $\Gamma^-(n)$ the set of nodes that follow $n$ (predecessors).

Each node produces a micro-blog piece of information, called post in the following. A post is defined as a sequence of terms $p=<t_0, t_1, t_2, \ldots, t_n>$. We denote by $P$ the set of posts and by $V_P$ the posts vocabulary.

*Filters*

To improve micro-blogging systems performance we propose keyword-based filters. A filter $F$ in our system is represented as a set of distinct terms $F = \{t_1, t_2, \ldots t_n\}$ where each term $t_i$ belongs to the filter vocabulary denoted by $V_F$ The length of $F$, denoted by $|F|$, is the total number of (distinct) terms it contains. Like Yan & Garcia-Molina (1994), we make the common assumption that $V_F \subseteq V_P$. $\mathcal{F}$ denotes the set of filters, excluding the filter $\perp$ that matches all posts, *i.e.*, $\perp = V_P$. A labeling function *label* associates a filter to each edge of the social graph $G$:

$$label: E \rightarrow \mathcal{F} \cup \perp$$

We name the social graph whose edges are labeled by filters the *filtered social graph* (*FSG*).

*Example.*

We present here an example of social graph that we will use throughout the paper to illustrate our different proposals.

Michel decides to follow two very active accounts, namely *CNN* and AFP (Agence France Press). These breaking news accounts publish dozens of posts every day about various topics. Since *AFP* is known as a reliable source with early breaking news, *CNN* follows *AFP* and retrieves all its posts. Conversely, AFP has a poorer cover in the IT and movies areas and relies on the posts issued by *CNN*, only for these two domains, *i.e.*, a small part of the posts published by *CNN*. To avoid flooding due to *CNN* and *AFP*, he would like to receive only posts concerning politics, IT and movies from *CNN* and only posts about politics from *AFP*. Cedric follows a very small number of accounts, including *AFP* from which he wants to receive sport news. Finally, Ryadh filters out the posts from *Cedric* to keep only those about *IT*. *Figure 1* illustrates the *FSG* corresponding to this motivating example.

Note that the filters are associated to the edges which allow a user to express different interests (*i.e.*, filters) w.r.t. the source considered. For instance the user *Michel* wants to retrieve all posts from *CNN* concerning *IT*, *politics* and *movies* and only these ones, and from *AFP* only posts about *politics*. Thus we have *label(Michel, CNN) = {IT, politics, movies}* and *label(Michel, AFP) = {politics}*.

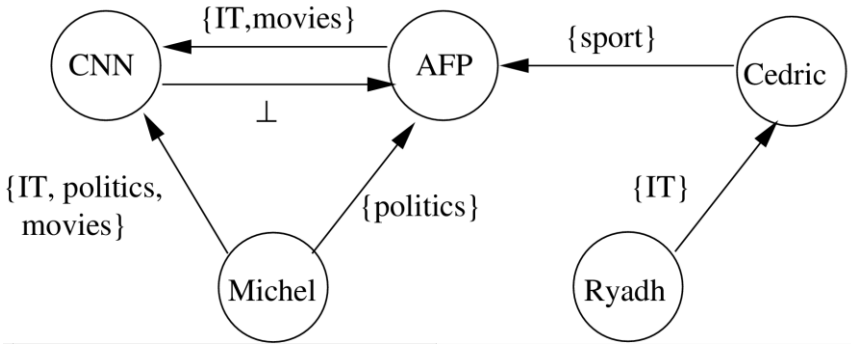


*Figure 1*. A filtered social graph

While currently no micro-blogging systems allow to filter out the posts received according to topics of interests, we believe this solution is an answer to the post flooding observed. We can envisage that these filters are explicitly expressed by the user or can be deduced from user interests. Regarding this second approach, we investigated several ways to automatically label the social graph edges with the corresponding filter.

Microblogging systems like *Twitter* provide hashtags which are a usage convention that allow users to tag a tweet by adding a # symbol before the tag. However if hashtags are a good

way to classify tweets, they have a very volatile cycle of life and are very often related to peculiar events. Moreover, most hashtags are formed by concatenation of words, or written in *SMS* language, what complicates their use if we want to introduce more complex matching (e.g. semantics). Additionally there are less than 10% of the tweets in our corpus that contain a hashtag which led us to discard this approach.

Teevan, Ramage, and Morris (2011) reported an average query size of *1.64* terms for the searches issued on the *Twitter* search engine. We assume an average filter size, extracted from the tweet content, similar to this average query size, motivated by *(i)* the filtering corresponds to a user preference regarding the publication of a given account and not general filters for all his interests on the collection of accounts he follows and *(ii)* we often meet in micro-blogging platforms users that are topical authorities (e.g., Pal & Counts, 2011), *Twitter* itself already recommends accounts to users for a given set of topics. With our system the users can specify filters to retrieve only the tweets related to the topic of authority of a user. Techniques such as topic modeling can also be used to infer relevant filters with an important extra-cost compared with our approach. In (Sommer et al., 2012), the authors describe a method which enables to label *tweets* with topics in order to provide sentiment analysis in an e-commerce context. We underline that our work aims at tackling scalability issues in such a context. For this purpose, we will not discuss the strategies (and their relevance) based on filter extraction from the tweet content like the one presented in Twitter's reference paper describing the search index (Busch et al., 2012). Improving the quality of filtering while remaining scalable is part of our future work.

## Indexing Schemes

The indexing scheme used in widespread micro-blogging systems like *Twitter* allows to efficiently retrieve for a publishing user *n* the set $\Gamma^{-}(n)$ of followers in a graph *G* without any filtering. These systems mainly rely on a hash-based index on the node *id* to determine the list of followers of this node. Our challenge when introducing filtering over more than 500 million users is how to efficiently determine the set $\Gamma^{-}(n)$ of followers based now on the *FSG*. This issue must be especially tackled for users with a large number of followers since the notification process time largely increases due to the containment relation to be checked, i.e. the length of the posting lists implies a time consuming scan to find users to notify.

We propose and compare different index structures that can easily extend the existing graph structure storage of online social network in order to manage post filtering. To achieve notification at runtime, regarding the high incoming rate of posts (*e.g., Twitter* reports some peaks with more than 7,000 tweets a second in 2011[13] and more than 140,000 tweets per seconds in 2013[14]), we consider structures that fit in memory. This discarded tree-based solutions. Our proposals are based on inverted lists which benefit from factorization and could be deployed on existing systems whose graph structure is already implemented as an inverted list. Note that using hash file implies that entries should have a collision resolution method such as classical separate chaining that uses pointers to an overflow space. Such a technique accommodates moderate growth, but to face the graph dynamicity we may need a dynamic hashing method such as linear hashing (Litwin, 1980).

Our variants of inverted lists exploit different factorizations: follower's ids, publisher's ids or term's. Since we effectively store terms' ids and not the terms themselves (we use a mapping

table to retrieve the term associated to an id) we consider in the following that all entries, follower's ids, publisher's ids or term's ids, require the same space (a *8-byte* integer in our implementation to fit Twitter's current *ids*). We denote by $\theta_{dir}$, $\theta_{list}$ and $\theta_{entry}$ the size of respectively a directory, list and entry (follower's, publisher's or term's ids).

Depending on the access order of the fields, we compare three different factorizations for our index structure. The *PFT-Index*, the *PTF-Index* and the *TPF-Index*. Each one of the proposed approach will be described the next subsections.

*The PFT-index*

The *PFT-index* (as *Publisher-Follower-Term* index) is essentially a mapping whose key is an account $n \in N$, and the value is the corresponding *posting list* $Postings_{PFT}(n)$, *i.e.*, the set of followers along with their filters: $Postings_{PFT}(n) = \left(n_1, \{t_{n_1}, t^2_{n_1}, \ldots\}\right)\left(n_1, \{t_{n_2}, t^2_{n_2}, \ldots\}\right)\ldots$, with $n_i \in \Gamma^-(n)$ and $t^j_{n_i} \in label(n_i, n)$. *PFT*-index corresponds to a factorization first on each publisher, and then for each publisher a second factorization on the followers 'IDs.

*Example*

*Figure 2* shows the PFT-index structure that corresponds to our Example.

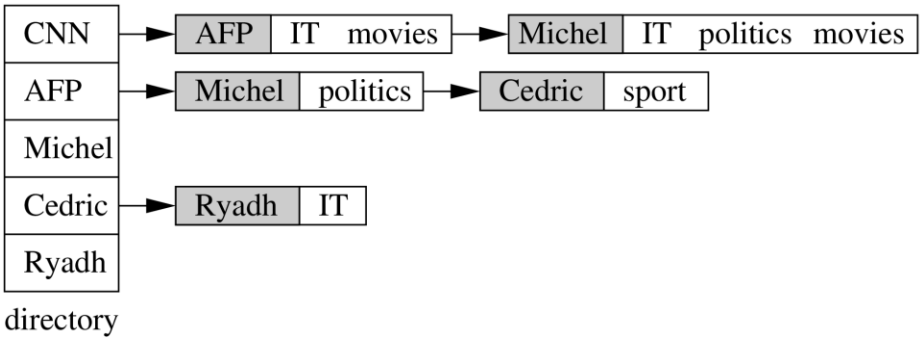


*Figure 2.* The PFT-index

*PFT-index memory requirement*

Let $\varphi$ be the average number of followers for a user, and $\tau$ the average number of filter terms for a (publisher, follower) pair. The index consists in the key directory, and the posting lists that contain followers 'IDs and terms. The expected memory requirement of *PFT-index* for a system represented by a graph *FSG* with $|N|$ users is:

$$\Delta^{PFT}_{memory}(FSG) = size(directory) + total(f\_id) + total(terms)$$

The size of the directory is the number of publishers, which we assume equals to $N$ (every account have generally at least one follower). The number of followers ids *total(f_id)* present in the structure is $N \times \varphi$, and the total number of terms *total(terms)* indexed is $N \times \varphi \times \tau$. We deduce:

$$\Delta^{PFT}_{memory}(FSG) = N \times \theta_{dir} + (N \times \varphi) \times \theta_{list} + (N \times \varphi \times \tau) \times \theta_{entry} \quad (1)$$

*PFT-index matching time*

Consider a post $p$ whose length is $|p|$ published by the user $n$. The notification process accesses the posting list $Postings_{PFT}(n)$ and for each follower $n_i$ it checks if it exists a term $t_j$ in post $p$ such that *label*($n_i$,$n$) contains $t_j$. Thus the expected average matching time is:

$$\Delta^{PFT}_{time}(FSG, p) = \varphi \times (|p| \times \tau) \quad (2)$$

*PFT-index insertion/deletion time*

To insert a new filter we scan the posting list $Postings_{PFT}(n)$ until we find the id of the follower that add/delete a filter $t$. Adding a new filter consists in appending $t$ to the corresponding term list. If the follower does not exist in $Postings_{PFT}(n)$, a new entry for this follower is added. Deleting requires an additional scan of the list of terms. Potentially it leads to the deletion of a follower's entry. Consequently the expected costs are respectively:

$$\Delta^{PFT}_{insert}(FSG) = \varphi / 2 \quad (3)$$

$$\Delta^{PFT}_{delete}(FSG) = \varphi / 2 + \tau / 2$$

*The PTF-index*

In the *PTF-index*, (as *Publisher-Term-Follower* index), a key is an account $n \in N$, and the value is the corresponding *posting list* $Postings_{PTF}(n)$. We factorize the posting list on the terms, so each term $t$ is associated to a list of the followers of $n$ that choose $t$ as a filter for the posts of $n$. So $Postings_{PTF}(n) = \{t_1, n^1_{t_1}, n^2_{t_1}, \ldots\}, \{t_2, n^1_{t_2}, n^2_{t_2}, \ldots\} \ldots$, with $n_i \in \Gamma^-(n)$ and $t^j_{n_i} \in label(n_i, n)$.

*Example*

*Figure 3* shows the PTF-index structure that corresponds to our Example.

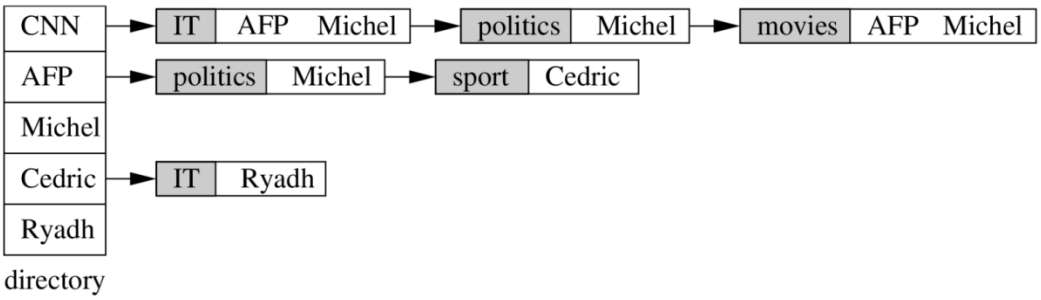


*Figure 3.* The PTF-index

*PTF-index memory requirement*

Our index consists of $N$ posting lists, each posting must store the $\varphi \times \tau$ filters associated to a publisher, like in *PFT-index*, with a factorization on the different terms. If we assume that all followers of a publisher use distinct filters, the size of $Postings_{PTF}(n)$ is $\varphi \times \tau$. However we observe that followers generally express similar interests when they decide to follow a given publisher and consequently the number of distinct filters for a publisher is lower than this upper bound. We assume like in many other text-based/keyword-based application that the total number of terms in a posting list follows a *Heaps' law* (Baeza-Yates & Ribeiro-Neto, 1999; Manning, Raghavan, & Schütze, 2008), *i.e.*, $|Postings_{PTF}(n)| = k \times T^{\beta}$, where $k$ and $\beta$ are constants and $T$ is the total number of terms in the posting. Heaps' coefficients $k$ and $\beta$ depend strongly on the characteristics of the analyzed text corpora and their value in our microblogging system has to be determined in future work. Note that $\beta$ is between 0 and 1 (generally in [0.4, 0.6]), so the higher the number of followers is, the better factorization is achieved. This is particularly expected in our filtering system where many users filter out on the same terms. Since the number of terms in $Postings_{PTF}(n)$ is $N \times (\varphi \times \tau)$ and the number of entries indexed is always $N \times \varphi \times \tau$, we deduce that:

$$\Delta^{PTF}_{memory}(FSG) = N \times \theta_{dir} + (N \times k \times (\varphi \times \tau)^{\beta}) \times \theta_{list} + (N \times \varphi \times \tau) \times \theta_{entry} \quad (4)$$

*PTF-index matching time*

Consider an incoming post $p$ published by the user $n$. We access the posting list of the publisher $n$ $Postings_{PTF}(n)$. Then for each entry $(t_i, n^1_{t_i}, \ldots, n^k_{t_i})$ we check if $p$ contains $t_i$. Whenever this happens we notify each $n^j_{t_i}$ from the entry $t_i$. Thus the expected average matching time is:

$$\Delta^{PTF}_{time}(FSG, p) = |p| \times k \times (\varphi \times \tau)^{\beta} \quad (5)$$

*PTF-index insertion/deletion time*

To insert a new filter $t$ we scan the posting list $Postings_{PTF}(n)$ until we find the entry that corresponds to $t$. Adding a new filter consists in appending the id of the follower that formulates this filter $t$ to the corresponding list. If the term does not exist in $Postings_{PTF}(n)$, a new entry for this term is added. Deleting requires an additional scan of the list of followers, on average $\frac{1}{2} \times \tau \times |Postings_{PTF}(n)|$. Potentially it leads to the deletion of a term's entry. Consequently the expected costs are respectively:

$$\Delta^{PTF}_{insert}(FSG) = k \times \frac{1}{2} \times \tau \times p / 2 \quad (6)$$

$$\Delta^{PTF}_{delete}(FSG) = k \times \frac{1}{2} \times \tau \times p / 2 + \frac{1}{2} \times \tau \times k \times \frac{1}{2} \times \tau \times p$$

*The TPF-index*

In the *TPF-index*, (as *Term-Publisher-Follower* index), the key is a term that appears in a filter. Thus the directory table contains the whole filter vocabulary $V_F$. The corresponding *posting list* $Postings_{TPF}(t)$ associated to a term $t$, is the set of publishers along with their followers that want to filter out this publisher on $t$:

$Postings_{tpf}(t) = \{n_1, n^1_{n_1}, n^2_{n_1}, \ldots\}\{n_2, n^1_{n_2}, n^2_{n_2}, \ldots\}\ldots$ with $n_i^j \in \Gamma^-(n_i)$ and $t \in label(n_i, n_i^j)$.

*TPF-index* corresponds to a factorization first on the term from $V_F$, and then for each term a second factorization on the publishers 'ids.

*Example*

*Figure 4* shows the TPF-index structure that corresponds to our Example.

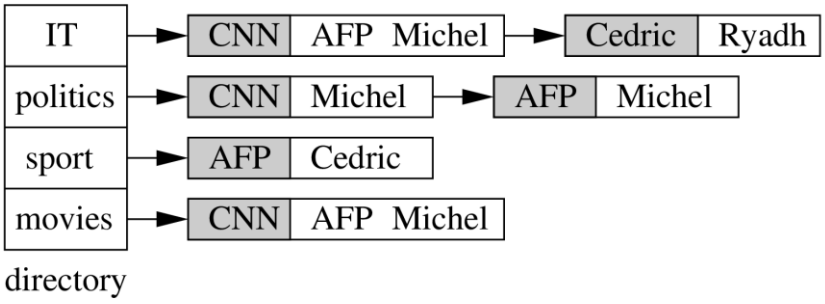


*Figure 4.* The TPF-index

*TPF-index memory requirement*

The *TPF-index* consists of $V_F$ posting lists, one for each term of the filter vocabulary. Since we have $N$ accounts that follow on average $\varphi$ other accounts with an average filter size of $\tau$, the total number of terms used for filtering is $N \times \varphi \times \tau$. We still assume that the size of the filter vocabulary follows a Heaps' law then the number of *distinct* terms is $|V_F| = k \times (N \times \varphi \times \tau)^{\beta}$. We make the common assumption that the distribution of terms in the set of filters follows the Zipf law (Baeza-Yates & Ribeiro-Neto, 1999; Manning, Raghavan, & Schütze, 2008), and that the number of publishers that are filtered out on a given term is proportional to the term frequency. Consequently, the number of publishers associated to a term $t_i$ whose frequency rank is $r_i$ is $\gamma / r_i$, where $\gamma$ is a constant. Here again $N \times \varphi \times \tau$ entries are finally stored, corresponding to the followers 'ids. Thus the expected memory requirement for the *TPF-index* is:

$$\Delta^{TPF}_{memory}(FSG) = |V_F| \times \theta_{dir} + \left( \sum_{i=1}^{|V_F|} \gamma / i \right) \times \theta_{list} + (|N| \times \varphi \times \tau) \times \theta_{entry}$$

Since $|V_F| >> 1$, we approximate $\sum_{i=1}^{|V_F|} \gamma / i$ with $\gamma \times \ln(|V_F|)$. Consequently we have:

$$\Delta^{TPF}_{memory}(FSG) = (k \times (N \times \varphi \times \tau)^{\beta}) \times \theta_{dir} + (\gamma \times \ln(k \times (N \times \varphi \times \tau)^{\beta})) \times \theta_{list} + (N \times \varphi \times \tau) \times \theta_{entry} \quad (7)$$

*TPF-index matching time*

Consider an incoming post $p$ published by the user $n$. For each term $t \in p$ we access the posting list of $t$ $Postings_{TPF}(t)$. Then for each entry $(n_i, n^1_{n_i}, n^2_{n_i}, \ldots)$ we check if $n_i$ is the publisher of $p$, *i.e.*, if $n_i = n$. If such an entry exists, we then notify each $n^j_{n_i}$ and stop the posting list scan for this term. Assuming the Zipf distribution of terms, the average posting list size is $\ln(|V_F|)/2$, and we scan on average half of the posting to find the publisher. Thus the expected average matching time is:

$$\Delta^{TPF}_{time}(FSG, p) = |p| \times (\gamma \times \ln(k \times (N \times \varphi \times \tau)^{\beta})/2)/2 \quad (8)$$

*TPF-index insertion/deletion time*

To insert a new filter $(n, n', t)$ we scan the posting list $Postings_{TPF}(t)$ whose size is on average $\ln(|V_F|)/2$ until we find the entry that corresponds to *n*. Adding a new filter consists in appending the id of the follower that formulates this filter *t* in the corresponding list. If the term does not exist in $Postings_{TPF}(t)$, a new entry is added. Deleting requires an additional scan of the list of followers, on average $N \times \varphi \times \tau / |V_F|$. Potentially it leads to the deletion of a publisher's entry. Consequently the expected costs are respectively:

$$\Delta^{TPF}_{insert}(FSG) = \lambda \times \ln(\kappa \times N \times \varphi \times \tau / |V_F|)/2 \cdot \mu/2 \quad (9)$$

$$\Delta^{TPF}_{delete}(FSG) = \lambda \times \ln(\kappa \times N \times \varphi \times \tau / |V_F|)/2 \cdot \mu/2 + N \times \varphi \times \tau / \mu |V_F|$$

## Experiments

In this section, we describe the experiments designed and performed in order to analyze *(i)* how our structures behave against real microblogging data from *Twitter* and *(ii)* the impact of different parameters of microblogging systems. These experiments validate our analytical model presented in the Indexing section.

*The dataset*

In collaboration with Forth Institute (Heraklion, Crete)[15] we gathered data on Twitter over a four month-period using the Twitter streaming API[16]. We generated a complete *Twitter graph + tweets* dataset by merging this data with the *Twitter* graph structure from Kwak, Lee, Park, & Moon (2010). As a result, we obtained a dataset with 2.9 million users, 169 million graph arcs and more than 33.9 million tweets. Finally, we performed a linguistic analysis for these tweets and kept only the English ones (and their associated accounts). Our resulting dataset is summarized in Table 2.

As explained earlier, we make the assumption that the average filter size (number of distinct terms that labeled an edge in the filtered social graph) corresponds to the average number of terms used on twitter querying API (Teevan, Ramage, & Morris, 2011). We decide for our experiments to generate filter terms for a follower through a draw among the most frequent and significant terms (we discarded urls, terms from the common language, Web shortcuts …) in the posts of the publisher s/he follows. Our rationale is that we usually follow a publisher because s/he provides some tweets that match one of our interests. Unless otherwise precised, this filter setting is used for our experiments. Remember that we do not intend in the current paper to validate the filter generation but to study scalability issues of such systems.

| *Element* | # |
|---|---|
| Users | 2,170,784 |
| Tweets | 15,717,449 |
| Graph arcs | 148,508,857 |

Table 2. *Dataset description*

| *User* | *Follower* | *Queries* |
|---|---|---|
| 12 | 36255503 | bigday twitter |
| 12 | 36255965 | conference deadline download |
| 12 | 36256156 | DB conference |
| 12 | 36256607 | software twitter |

Table 3. *Filtered social graph dataset sample*

Table 3 shows a sample of the filter social graph obtained using our label generation process. Experiments run on an Intel Core i5 CPU with a frequency of 3.60 GHz and 16 GB of RAM. The structures are implemented in JAVA.

*Memory requirement*

All structures have different factorization criteria which lead to different memory requirements. Table 4 compares the space occupancy of the three structures for our dataset. *TPF-index* appears as the structure with the lower memory requirements. Many filters are shared by a significant number of users which allows a better factorization, on the terms first. Moreover, we observe that many followers of a publisher filter out on the same terms. Consequently, for a term's entry in the *TPF-index*, there exists also an important factorization on publisher's id, especially for account with an important number of followers. Oppositely the *PFT-index* benefits from a poor factorization since all publishers have an entry in the directory and for each of them we have a list element for each of his followers, each of them with few filter terms.

| *Structure* | *Index size (MB)* |
|---|---|
| PFT | 9021 |
| PTF | 3777 |
| TPF | 1869 |

Table 4. *Index sizes for the realistic dataset*

To measure the impact of the different parameters of our microblogging system and to validate our analytical model, we generate synthetic datasets with a constant number of filters ($\tau$) for each graph edge. We report results in *Figure 5*. The memory occupancy grows linearly with $\tau$ for *PFT* and *PTF* indexes. Indeed, increasing $\tau$ does not impact the directory size that depends only on the number of publishers, *i.e.*, $N$. Moreover, for *PFT-index*, the number of elements in the posting list remains constant and equal to the number of followers. But the number of entries follows linearly $\tau$. For *PTF-index* the number of elements depends on the number of distinct term used as filter for a publisher. When comparing with *PFT-index* we observe the same gradient.

This reveals that τ has a low impact on the number of elements for a posting list in *PTF-index*. The Heaps' law we propose in our model explains this result, since it assumes that the sub-vocabulary of filter terms for a given publisher increases slightly. Thus for *PTF-index*, like for *PFT-index*, the increase of $\tau$ does not impact the structure but only the number of entries. Finally, we note the low impact of $\tau$ on the *TPF-index*. The rationale is *(i)* the directory size remains constant and equal to $V_F$, *(ii)* since filters are generated w.r.t. the publisher's post areas, a new filter term has a high probability to be already present for the same publisher in the structure, so the number of posting elements remains low and slowly increases.

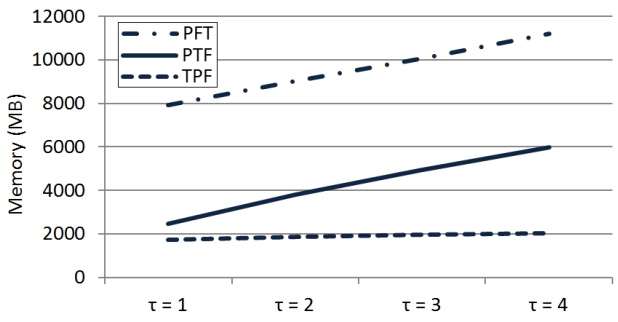


*Figure 5.* Occupied memory w.r.t. τ

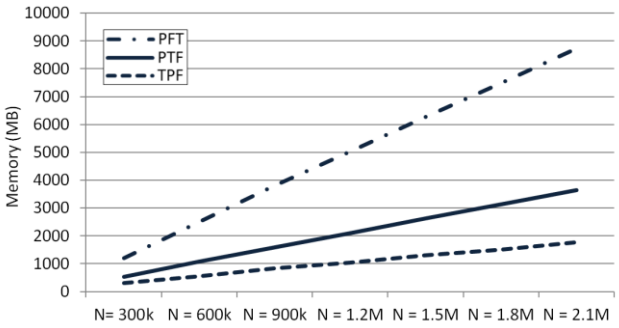


*Figure 6.* Occupied memory w.r.t. *N*

*Figure 6* illustrates the impact of *N* on the different structures. All structures exhibit a linear growth. For *PFT* and *PTF* index, adding a new user results in adding a new directory entry, new posting list elements and new posting entries for his followers along with their filters. However the factorization on terms' id in a posting list is more efficient than the one on follower's id which explains the best gradient for *PTF*. For *TPF* after a short initialization step that corresponds to the creation of the different entries of the directory and the different elements of the posting list, increasing *N* leads only to add new posting entries. This explains a linear growth with a lower gradient. Both Figures validate our analytical model.

*Indexing times*

We compare in Table 5 the time needed to build the different structures in central memory for different datasets. Uniform (1), (2) and (3) correspond to scenarii where each publisher-follower edge is labeled by respectively 1, 2 or 3 filter terms. As expected, the smaller index size is, the lower building time we have. Indeed for *TPF-index* we have smaller posting lists, so we need less time to find the list element where the new filter must be inserted. Oppositely in *PFT-index*, a time-consuming scan of large posting list is required.

We also note that the building time is not proportional to the value of $\tau$. It turns out that posting lists elements are quickly created and their number slowly evolves. After this step, increasing the number of filter terms for a follower corresponds mainly to add new entries in existing elements which corresponds to a (almost) constant time, explaining the linear behavior. These results are in accordance with our analytical model.

| *Structure* | *realistic* | *uniform(1)* | *uniform(2)* | *uniform(3)* |
|---|---|---|---|---|
| PFT | 753 | 736 | 741 | 1081 |
| PTF | 512 | 357 | 444 | 558 |
| TPF | 231 | 198 | 236 | 289 |

Table 5. *Indexing times (in s)*

*Matching times*

To evaluate matching time we process as follows: first, the post is decomposed into a bag of terms, then we use the index to determine for each term of the post the set of followers to be notified. Observe that since we are working in a disjunctive logic, when the first positive match occurs we can directly notify the corresponding user. Table 6 depicts average matching times for an upcoming flow of 100,000 tweets over the different filter indexes.

We observe that the *TPF-index* exhibits poor matching performances: for our realistic dataset, with around 2.5 ms a post, it can handle less than 400 posts a second, so far from being scalable (remember that in Twitter for instance there exist peaks with 8,000 posts a second). For each term of the post we retrieve a large posting list with potentially as many elements as existing publishers $N$. Oppositely *PTF-index* quickly retrieves the followers to be notified: it handles a post in 15µs, so is able to manage peaks up to 66K posts a second. Here we directly access the posting list corresponding to the publisher. Then we scan all its elements that correspond to all followers of this account, and check for each of them if any filter term matches the post. Observe that the number of filter terms is low (between 1 and 3) for a follower, and that we check all terms of the post in a single scan. Of course this is an average value and the matching time is higher for publisher with numerous followers and faster for those with few ones.

| *Structure* | PFT | PTF | TPF |
|---|---|---|---|
| *matching time (µs)* | 808 | 15 | 2564 |

Table 6. *Matching time for realistic dataset*

*Figure 7* illustrates the impact of $\tau$ on matching time. Like in Table 6, *PTF-index* outperforms other proposals with 2 orders of magnitude. We observe that the matching time for *PFT-index* linearly increases with $\tau$ while *TPF-index* follows a sub-linear growth. For the former, matching implies a direct access to the posting list of a publisher and then to scan all elements in turn for this list to check if associated entries match the post terms. Increasing $\tau$ does not change neither the number of entries of the directory, nor the number of elements. So only the last step, the matching attempt against term entries requires more time. Since the number of entries is proportional to $\tau$, this explains this linear growth.

For the *TPF-index*, we observe the Zipf's law behavior in the term frequency distribution, so when increasing $\tau$ we generally add entries in the posting of the most frequent terms. As a consequence the more numerous filters are indexed, the higher probability we have to add an entry in an existing posting list element. Since posting lists' size has a sub-linear increase and since we scan as many lists as number of terms in the post, this justifies that the matching time increases sub-linearly w.r.t. $\tau$.

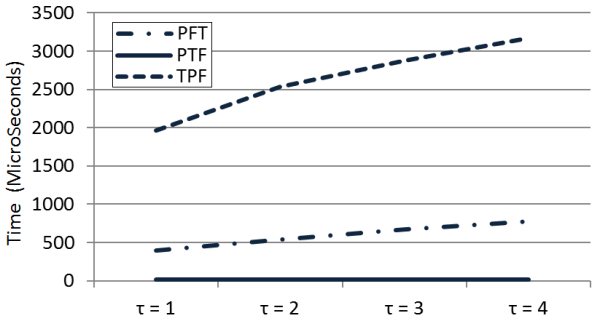


*Figure 7*. Matching time w.r.t. τ

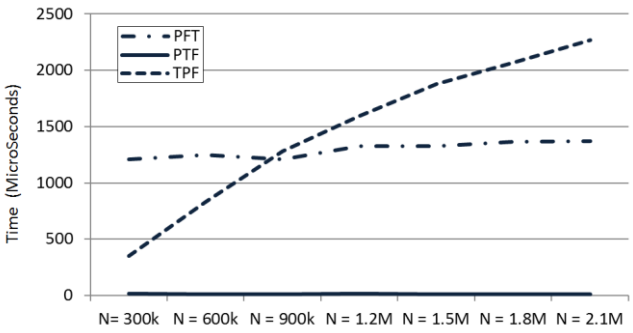


*Figure 8*. Matching time w.r.t. *N*

We report also in *Figure 8* the evolution of matching time w.r.t. *N*. We notice that both *PFT* and *PTF-index* have a constant matching time. Indeed, increasing *N* only impact the directory by adding new directory entries, but the posting lists keep a constant number of elements and for each element a constant number of entries. Since for an incoming post we scan a single posting list corresponding to the publisher, the matching time is constant with *N*. *TPF-index* exhibits better performance than *PFT-index* for *N* lower than *900k*. The matching time with *TPF-index* increases sub-linearly w.r.t. *N* for the same reasons as with $\tau$. These results also confirm our analytical model.

*Filtering efficiency*

To evaluate how filtering reduces the number of posts delivered, we compare the number of posts that are sent from the microblog server to the users with and without filtering. As expected the number of posts delivered dramatically drops since we measure a gain of almost 98% for all our datasets (see Table 7). Our disjunctive hypothesis for filters justifies that the more numerous filter terms we have, the more numerous *posts* have to be delivered. However observe that we do not have a linear increase in the number of delivered posts. Indeed, there is a higher probability for a post to match several filter terms of a follower when followers have large filters.

| *Filtering* | *Realistic* | *uniform(1)* | *uniform(2)* | *uniform(3)* | *uniform(4)* |
|---|---|---|---|---|---|
| *Enabled* | 211,651 | 161,196 | 212,364 | 244,578 | 267,164 |
| *Disabled* | 11,035,437 | | | | |

Table 7. *Number of delivered posts*

*Graph evolution*

Users in microblogging systems have a dynamic following behavior (Kwak et al. 2011; Kivran-Swaine et al. 2011). The underlying social graph in *Twitter*-like networks is in a constant evolution. Relationships are subject to move because they depend on various fluctuating factors like temporal interest for an event and accounts related to this event, publisher's content creation rates, etc. To illustrate how the *PFT*, *PTF*, *TPF* and *Hybrid* (presented below) structures handle the graph dynamicity we reported average insertion/deletion times in Table 8.

Since the TPF-index do not reflect the graph structure, a time consuming scan of the lists have to be performed for popular filters in the case of insertion as well as for popular accounts entries in the case of deletion. Also, as expected from the analytical study, the PFT-index is the most dynamic structure and requires 30% less time for inserting and 67% less time for deleting compared to the PTF. The explanation lies in the structure of the PFT whose factorization, first on the publisher, then on the follower, corresponds to an edge factorization. Thus adding or removing an edge is performed fast. The factorization by term in the PTF-index broke the natural graph storing leading to worst update times.

| *Structure* | *Avg. Insertion time* | *Avg. Deletion time* |
|---|---|---|
| *PFT-index* | 2274 | 566 |
| *PTF-index* | 3247 | 1731 |
| *TPF-index* | 3023 | 1057 |
| *Hybrid-index (threshold = 400)* | 2498 | 818 |

Table 8. *Avg. insertion/deletion times (in Nano-Seconds)*

*The hybrid structure*

Like many other social networks, microblogging systems are characterized by the heterogeneity of the accounts. *Figure 9* and *Figure 10* illustrate this heterogeneity in our dataset. We see in *Figure 9* that we can distinguish 5 classes of users based on their number of followers. We observe in *Figure 9* that more than half of the users (55.78%) are followed by less than ten users. At the same time we have in the system a minority of very popular accounts that are followed by a large number of accounts (0.78% are followed by more than 1,000 users).

Regarding the publication activity, we observe in *Figure 10*, that the classes with the highest number of followers have also the highest publication rates. This is a common topological phenomenon in social networks also described by Kwak, Lee, Park, & Moon (2010) and Ahn, Han, Kwak, Moon, & Jeong (2007).

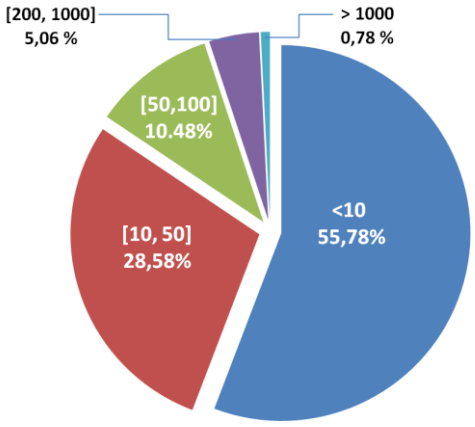


*Figure 9.* Users by their number of followers

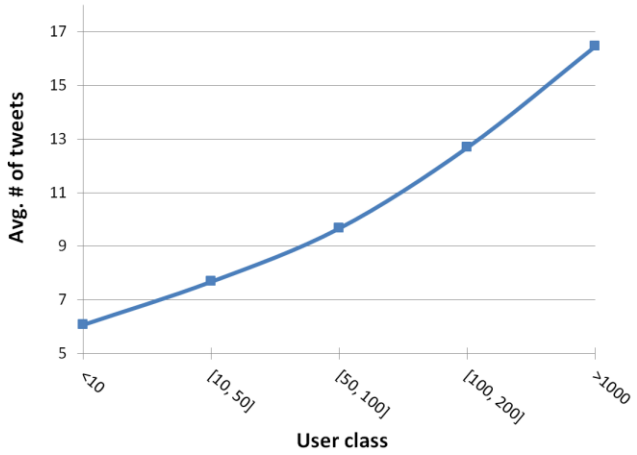


*Figure 10.* Users classes by their average number of tweets

Given this specificity, we investigated an hybrid structure that combines both *PFT* and the *PTF* index structures. Basically this structure stores the accounts that have a number of followers lower a certain threshold similarly to the *PFT* structure and other accounts using a *PTF* factorization. The main motivation behind this choice is that *PTF* index handles better the popular publishers than the *PFT* one (see Section "Matching times").

*Hybrid structure memory requirements*

The key parameter in the hybrid structure is the value chosen for the threshold, which represents the minimal number of followers we may have to be considered as a popular account. We denote this threshold as the *popularity threshold* in the following. According to this value we decide whether we store an account as a *PFT* (not popular) or as a *PTF* (popular) index entry.

Our results for the memory requirement regarding the popularity threshold value are presented in *Figure 11*. We see that handling larger accounts (in terms of number of followers) as PTF have a direct impact on the index sizes. The smaller the popularity threshold is set, the smaller is the size of the structure. The size of the hybrid structure lies consequently between the PFT and the PTF. For instance a value of 5 for the popularity threshold lead to a memory requirement similar to PTF while a popularity threshold of 400 leads to an increase of 30% of the memory requirements.

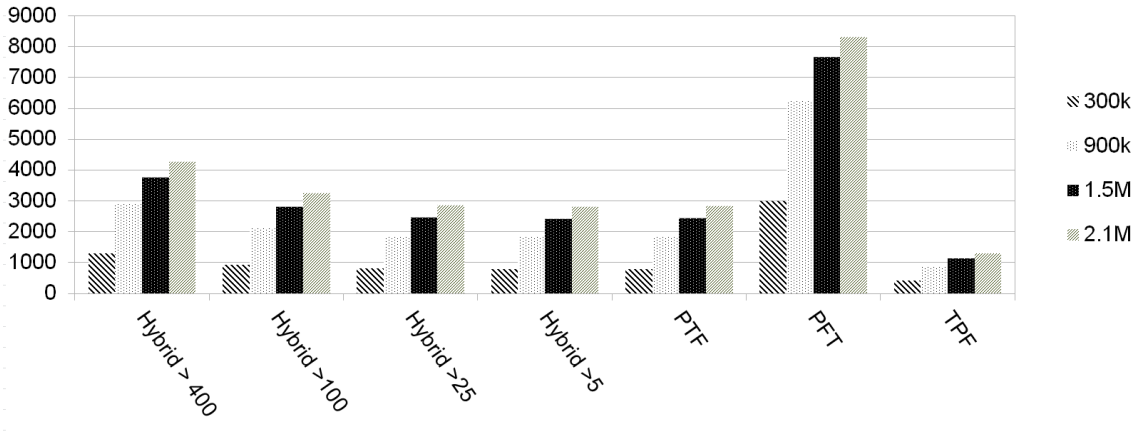


*Figure 11*. Occupied memory w.r.t. N and hybrid threshold value

*Hybrid structure matching times*

We report the matching times obtained with the hybrid structure in Figure 12. We observe that the matching time remains constant when varying the popularity threshold from 5 to 25. With this users partitioning parameter, the hybrid structure performs a matching in 15µs so similar to the PTF (see Section "Matching times"). Table 9 shows the number of users (on the 2.17M in our graph) indexed as PTF for various popularity thresholds. We see that starting from the popularity threshold 25, which means that almost 25% of the users are considered as popular account, the matching time quickly increases. However we observe that with a popularity threshold of 400, only 2.5% of the accounts are considered as popular and are indexed as a PTF entry, but the gain compared to the full PFT indexing is 25.2 (32µs versus 808µs). This confirms our motivation to distinguish the (rare) popular accounts, and the others when indexing.

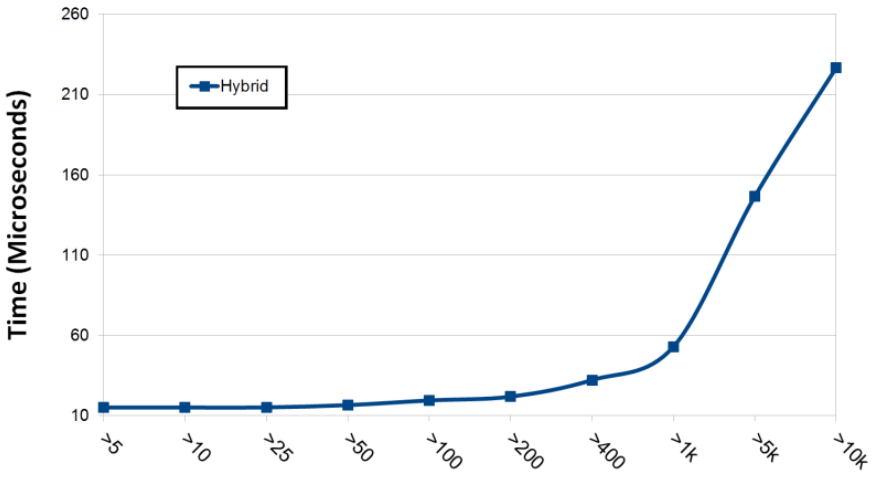


*Figure 12*. Matching times for the hybrid structure w.r.t popularity threshold

| *Popularity threshold* | *Number of users* |
|---|---|
| 5 | 1,349,490 |
| 10 | 959,976 |
| 25 | 547,001 |
| 50 | 337,393 |
| 100 | 199,033 |
| 200 | 209,844 |
| 400 | 54,024 |
| 1000 | 16,961 |
| 5000 | 2,118 |
| 10000 | 896 |

Table 9. Number of users for various popularity thresholds

Regarding the dynamicity of the structures, the hybrid structure handles update of the social graph and/or filter more efficiently than PTF, and almost as fast as the *PFT* (see Table 8): an insertion is performed in 2498 *ns* and a deletion in 818 *ns*, versus respectively 2274 *ns* and 566 *ns* for *PFT* for a popularity threshold set to 400. Consequently we conclude that handling larger accounts (in number of followers) with a specific factorization can improve the overall efficiency of the system. The hybrid structure achieves an interesting compromise between *PTF* and *PFT*. For instance with a popularity threshold of 400, we need 30% more space than *PTF* but 50% less than *PFT*. The matching time is twice the one of *PTF*, but 25 times less than *PFT*.

However, oppositely, the hybrid structure exhibits better performances than *PFT* regarding the management of dynamicity of the graph with a gain of 23% for insertion time and 53% for deletion time.

## Conclusion and future work

In the present paper we compare inverted lists-based structures that index filters to decrease the number of messages delivered in microblogging systems. We propose an analytical model for all these structures and validate them with real and synthetic datasets. *PTF-index* appears to achieve the best scalability since, despite the fact that it require more memory and insertion time than the *TPF-index*, it outperformed with two orders of magnitude other proposals for matching time. We have also shown that exploiting the heterogeneity of the accounts (*e.g.* 5% of accounts with more than 100,000 followers) as we did with the hybrid structure, allows to achieve better overall performance especially when considering the graph evolution and dynamicity.

We intend in future work to consider other optimizations like clustering or summarization which group different filters inside a posting list to achieve better performance. Adding

conjunction and negation in filter expressions as well as more complex matching schemes is another future challenge.

Finally, we are currently working on real-time content recommendation for micro-blogging systems. The idea is to rely on our time and memory efficient indexing structures combined with semantic distances to provide user recommendations on top of the *FSG*.

*Acknowledgement*

This work has started in collaboration with Michel Scholl who passed away the 15th of November 2011, too early. The authors would like to thank Michel Scholl for his devotion to the database research.

## References

Ahn, Y.-Y., Han, S., Kwak, H., Moon, S. B., & Jeong, H. (2007). Analysis of topological characteristics of huge online social networking services. In *Proc. Intl. World Wide Web Conference (WWW)* (pp. 835–844).

Baeza-Yates, R. A., & Ribeiro-Neto, B. A. (1999). *Modern Information Retrieval*. ACM Press / Addison-Wesley.

Bakshy, E., Hofman, J. M., Mason, W. A., & Watts, D. J. (2011). Everyone's an Influencer: Quantifying Influence on Twitter. In *Proc. Intl. Conf. on Web Search and Web Data Mining (WSDM)* (pp. 65–74).

Bontcheva, K., Gorrell, G., & Wessels, B. (2013). Social Media and Information Overload: Survey Results. *arXiv:1306.0813 [cs.SI]*. Retrieved from http://arxiv.org/abs/1306.0813

Bouraga, S., Jureta, I., Faulkner, S., & Herssens, C. (2014). Knowledge-Based Recommendation Systems: A Survey. In *International Journal of Intelligent Information Technologies (IJIIT),* 10(2), 1–19.

Busch, M., Gade, K., Larson, B., Lok, P., Luckenbill, S., & Lin, J. (2012). Earlybird: Real-Time Search at Twitter. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)* (Vol. 0, pp. 1360–1369).

Broder, A. Z., Das, S., Fontoura, M., Ghosh, B., Josifovski, V., Shanmugasundaram, J., & Vassilvitskii, S. (2011). Efficiently Evaluating Graph Constraints in Content-Based Publish/Subscribe. In *Proc. Intl. World Wide Web Conference (WWW)* (pp. 497–506).

Chen, C., Li, F., Ooi, B. C., & Wu, S. (2011). TI: An Efficient Indexing Mechanism for Real-time Search on Tweets. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data* (pp. 649–660).

Foster, J., Çetinoğlu, Ö., Wagner, J., Roux, J. L., Hogan, S., Nivre, J., … Genabith, J. van. (2011). #hardtoparse: POS Tagging and Parsing the Twitterverse. In *Proc. Intl. Work. on Analyzing Microtext (AMW)*.

Haghani, P., Michel, S., & Aberer, K. (2010). The Gist of Everything New: Personalized Top-k Processing over Web 2.0 Streams. In *Proc. Intl. Conf. on Information and Knowledge Management (CIKM)* (pp. 489–498).

Hmedeh, Z., Kourdounakis, H., Christophides, V., Mouza, C. du, Scholl, M., & Travers, N. (2012). Subscription Indexes for Web Syndication Systems. In *Proc. Intl. Conf. on Extending Database Technology (EDBT)* (pp. 1–12).

Hmedeh, Z., Vouzoukidou, N., Travers, N., Christophides, V., Mouza, C. du, & Scholl, M. (2011). Characterizing Web Syndication Behavior and Content. In *Proc. Intl. Conf. onWeb Information Systems Engineering (WISE)* (pp. 1–12).

Java, A., Song, X., Finin, T., & Tseng, B. L. (2007). Why We Twitter: An Analysis of a Microblogging Community. In *Proc. Intl. Work. on Advances in Web Mining and Web Usage Analysis (SNA-KDD)* (pp. 118–138).

Kivran-Swaine, F., Govindan, P., & Naaman, M. (2011). The Impact of Network Structure on Breaking Ties in Online Social Networks: Unfollowing on Twitter. In *Proc. Intl. Conf. on Human Factors in Computing Systems (CHI)* (pp. 1101–1104).

König, A. C., Church, K. W., & Markov, M. (2009). A Data Structure for Sponsored Search. In *Proc. Intl. Conf. on Data Engineering (ICDE)* (pp. 90–101).

Kwak, H., Chun, H., & Moon, S. B. (2011). Fragile Online Relationship: A First Look at Unfollow Dynamics in Twitter. In *Proc. Intl. Conf. on Human Factors in Computing Systems (CHI)* (pp. 1091–1100).

Kwak, H., Lee, C., Park, H., & Moon, S. B. (2010). What Is Twitter, a Social Network or a News Media? In *Proc. Intl. World Wide Web Conference (WWW)* (pp. 591–600).

Levering, R., & Cutler, M. (2006). The portrait of a common HTML web page. In *ACM Symp. on Document Engineering* (pp. 198–204).

Liang, H., Xu, Y., Tjondronegoro, D., & Christen, P. (2012). Time-aware topic recommendation based on micro-blogs. In *Proc. Intl. Conf. on Information and Knowledge Management (CIKM)* (pp. 1657–1661).

Litwin, W. (1980). Linear Hashing: A New Tool for File and Table Addressing. In *Proc. Intl. Conf. on Very Large Data Bases (VLDB)* (pp. 212–223).

Ma, S., & Zhang, Q. (2007). A Study on Content and Management Style of Corporate Blogs. In *HCI (15)* (pp. 116–123).

Manning, C. D., Raghavan, P., & Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press.

Pal, A., & Counts, S. (2011). Identifying Topical Authorities in Microblogs. In *Proc. Intl. Conf. on Web Search and Web Data Mining (WSDM)* (pp. 45–54).

Pereira, J., Fabret, F., Llirbat, F., Preotiuc-Pietro, R., Ross, K. A., & Shasha, D. (2000). Publish/Subscribe on the Web at Extreme Speed. In *Proc. Intl. Conf. on Very Large Data Bases (VLDB)* (pp. 627–630).

Sankaranarayanan, J., Samet, H., Teitler, B. E., Lieberman, M. D., & Sperling, J. (2009). TwitterStand: News in Tweets. In *Proc. Intl. Symp. on Geographic Information Systems (ACM-GIS)* (pp. 42–51).

Silberstein, A., Terrace, J., Cooper, B. F., & Ramakrishnan, R. (2010). Feeding Frenzy: Selectively Materializing Users' Event Feeds. In *Proc. Intl. Conf. on Management of Data (SIGMOD)* (pp. 831–842).

Sommer, S., Schieber, A., Heinrich, K., & Hilbert, A. (2012). What is the Conversation About?: A Topic-Model-Based Approach for Analyzing Customer Sentiments in Twitter. In *International Journal of Intelligent Information Technologies (IJIIT)*, 8(1), 10–25.

Sriram, B., Fuhry, D., Demir, E., Ferhatosmanoglu, H., & Demirbas, M. (2010). Short Text Classification in Twitter to Improve Information Filtering. In *Proc. Intl. Conf. on Research and Development in Information Retrieval (SIGIR)* (pp. 841–842).

Teevan, J., Ramage, D., & Morris, M. R. (2011). #TwitterSearch: a Comparison of Microblog Search and Web Search. In *Proc. Intl. Conf. on Web Search and Web Data Mining (WSDM)* (pp. 35–44).

Uysal, I., & Croft, W. B. (2011). User Oriented Tweet Ranking: A Filtering Approach to Microblogs. In *Proc. Intl. Conf. on Information and Knowledge Management (CIKM)* (pp. 2261–2264).

Vosecky, J., Leung, K. W.-T., & Ng, W. (2012). Searching for Quality Microblog Posts: Filtering and Ranking Based on Content Analysis and Implicit Links. In *Proc. Intl. Conf. on Database Systems for Advanced Applications (DASFAA)* (pp. 397–413).

Weng, J., Lim, E.-P., Jiang, J., & He, Q. (2010). TwitterRank: Finding Topic-sensitive Influential Twitterers. In *Proc. Intl. Conf. on Web Search and Web Data Mining (WSDM)* (pp. 261–270).

Wu, L., Lin, W., Xiao, X., & Xu, Y. (2013). LSII: An indexing structure for exact real-time search on microblogs. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)* (Vol. 0, pp. 482–493).

Yan, T. W., & Garcia-Molina, H. (1994). Index Structures for Selective Dissemination of Information Under the Boolean Model. *ACM Transactions on Database Systems (TODS), 19*(2), 332–364.

Yin, H., Li, J., & Niu, Y. (2014). Detecting Local Communities within a Large Scale Social Network Using Mapreduce. In *International Journal of Intelligent Information Technologies (IJIIT)*, 10(1), 57–76.

Zhang, Y., Li, X., & Wang, T.-W. (2013). Identifying Influencers in Online Social Networks: The Role of Tie Strength. In *International Journal of Intelligent Information Technologies (IJIIT*, 9(1), 1–20.

Figure Captions

## Footnotes

[1] http://www.twitter.com
[2] http://www.telegraph.co.uk/technology/9837525/Half-a-billion-people-sign-up-for-Twitter.html
[3] https://about.twitter.com/company
[4] http://www.weibo.com
[5] http://www.identi.ca
[6] http://www.plurk.com
[7] http://latimesblogs.latimes.com/technology/2009/02/twitter-creator.html
[8] https://twitter.com/FoxNews
[9] https://twitter.com/BarackObama
[10] http://fr.slideshare.net/adityabheemarao/twitter-architecture-and-scalability-lessons
[11] http://www.infoq.com/presentations/Twitter-Timeline-Scalability
[12] http://latimesblogs.latimes.com/technology/2011/07/twitter-delivers-350-billion-tweets-a-day.html
[13] http://yearinreview.twitter.com/en/tps.html
[14] https://blog.twitter.com/2013/new-tweets-per-second-record-and-how
[15] We thank Vassilis Christophides for his support and helpful comments
[16] http://dev.twitter.com/