Efficient Filtering in Micro-blogging Systems: We Won't Get Flooded Again

Ryadh Dahimene, Cedric du Mouza, and Michel Scholl

CEDRIC Laboratory - CNAM - Paris, France
firstname.lastname@cnam.fr

Abstract. In the last years, micro-blogging systems have encountered a large success. Twitter for instance claims more than 200 million accounts after 5 years of existence with more than 200 million tweets a day leading to 350 billion delivered tweets. Micro-blogging 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 his list of postings to read news that match his interests. Meanwhile these systems receive all posts and deliver each of them to *all* the followers of the publishing accounts, whether they are interested by the news or not. To avoid user flooding and to significantly diminish the number of posts to be delivered, we propose in this paper three filtering structures for micro-blogging systems. They allow to efficiently retrieve the followers of an account that could be interested by a post s/he published. We compare analytically these structures and confirm our analysis experimentally on synthetical datasets and on a real Twitter dataset which consists of more than 2.1 million users, 15.7 million tweets and 148.5 million publisher-follower relationships.

Keywords: Micro-Blogging, Filtering, Indexing

1 Introduction

Micro-blogging systems have become a major trend over the Web 2.0 as well as an important communication vector. In less than six years, *Twitter*[1] has grown in a spectacular manner to reach more than 200 million active users in August, 2011[2]. Other similar systems like *Sina Weibo*[3], *Identi.ca*[4] or *Plurk*[5], 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 generally limited to 140 characters (14.7 terms [5]), so clearly greater than bids (4-5 terms [11]) but smaller

[1] http://www.twitter.com
[2] http://www.jeffbullas.com/2011/09/21/11-new-twitter-facts-figures-and-growth-statistics-plus-infographic/
[3] http://www.weibo.com
[4] http://www.identi.ca
[5] http://www.plurk.com

than RSS items [8], blogs (250-300 terms [16]) or Web pages (450-500 terms excluding tags [15]). This last point is considered to be one the key factors of success for micro-blogging systems.

In micro-blogging, 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 him, which results in the existence of a large social graph. Micro-blogging is characterized by the heterogeneous nature of the accounts. In *Twitter* for instance, there exist some high update frequency accounts (newspapers, tech-blogs, journalists) and others that publish less than one post a week. Moreover there exist very popular accounts (*e.g.* High-tech blog account like @techcrunch or celebrity accounts like @ladygaga with more than 16.8 million followers), and others with 0 or 1 follower.

For various reasons (security, advertisement, control policy, ...) these systems rely on a centralized architecture. Each post published is received by the system that forwards it to *all* followers of the publishing account. Since 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 claims more than 200 million tweets a day, must deliver daily over 350 billion tweets[6]. This traffic overload (especially for high-followed accounts) represent 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 (see [13] for Twitter), lets him lost in the middle of long feed of posts. This results in poor data readability and potentially loss of valuable information. A direct consequence of this phenomenon is the high dynamicity of the graph: to avoid flooding, users follow temporary an account to cover a peculiar event, and then unsubscribe because they can't manage the continuous flow of posts [12] [10].

In order to improve the user experience and reduce the network load, we have chosen to introduce filtering in micro-blogging 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, expressed as keywords. To scale, the 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 specific aspects of micro-blogging systems which we can summarize as:

- **short messages:** the size restriction (generally 140 characters) means that we handle short documents (*e.g.* the average length of a *tweet* is 14.7 terms [5]);
- **account heterogeneity:** micro-blogging studies have revealed account heterogeneity in term of both update frequency and number of followers;
- **graph evolution:** As observed in Twitter [10, 12], users follow and unsubscribe often to other accounts. The filtering structure must consequently be dynamic to face this phenomenon;

[6] http://latimesblogs.latimes.com/technology/2011/07/twitter-delivers-350-billion-tweets-a-day.html

– **centralized system:** The social graph is stored by the micro-blogging 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.

This paper is, as far as we know, the first attempt to propose such a structure that combines the social graph and keyword-based filters in a micro-blogging system, considering the main characteristics of such a system.

Motivating example

We present here an example of social graph that we will use throughout the paper to illustrate our different proposals. Michel decide 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 follows. For instance he filters out the posts from *Cedric* to keep only those about IT (see Figure 1).

The rest of the paper presents in Section 2 the data model, including the formalization of the filters and of the notification. Section 3 describes our three filtering structures along their analytical study. Section 4 presents our experimental setting and results. Related work follows in Section 5 and Section 6 concludes the paper.

2 Data model

In this Section we introduce our micro-blogging model and our content filtering based on keywords queries.

2.1 Micro-blogging system

Basically a micro-blogging system like *Twitter* can be represented as a directed graph *G=(N, E)* where the set of nodes N represent the users (accounts) of the system and the set of edges $E \subseteq N \times N$ represent 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).

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

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

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

Each node produces micro-blog piece of information, called *post* in the following. A post is defined as a sequence of terms $p =< t_0, t_1, t_2, \ldots, t_i >$. We denote by P the set of posts and by $\mathcal{V}_P$ the posts vocabulary. Note that we do not rely on the terms order for matching (see Section 2.3) and consequently we consider the bag of terms of the post $p = \{t_0, t_1, t_2, \ldots, t_i\}$ rather than the term sequence in the following. The *posts sequence* of a node n, denoted n_s refers to the the chronologically sorted sequence of posts $n_s =< p_0, p_1, p_2, \ldots, p_i >$, published by n.

2.2 Filters

To improve micro-blogging systems performances we propose keyword-based filters. A filter F in our system is represented as a set of distinct terms $F = \{t_1, t_2, ..., t_n\}$ where each term t_i belongs to the filter vocabulary denoted by $\mathcal{V}_F$. The length of F, denoted by $|F|$, is the total number of (distinct) terms it contains. Like [25], we make the common assumption that $\mathcal{V}_F \subseteq \mathcal{V}_P$. $\mathcal{F}$ denotes the set of filters, excluding the filter $\perp$ that matches all posts, *i.e.*, $\perp = \mathcal{V}_P$. 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*). Note that the filters are associated to the edges which allows 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 illustrates the FSG for the example given in Section 1.

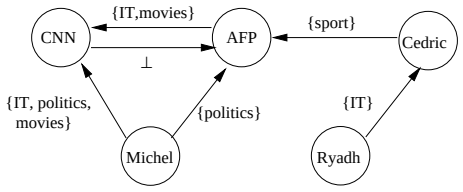

Fig. 1. A filtered social graph

2.3 Notification

We consider a *disjunctive logic* for the notification: the user is notified when s/he follows an account that publish a post which matches at least one of the terms of his filter. Extensions to introduce conjunctions or negations are part of future work. Formally, we define the notification as follows.

Definition 1 (Notification). *The set of nodes $\mathcal{N}$ to be notified for an incoming post p at node n is:*

$$\mathcal{N}(n,p) = \{n' \in N | (n',n) \in E, \exists t \in p, t \in label(n',n)\}$$

3 Filter indexing

Currently 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 graph G. These systems mainly rely on a hash-based index on the node *id* that allows to retrieve the list of followers of this node (see [1] for a Twitter architecture presentation for instance). [21] reports an average query size of 1.64 terms for the searches issued on the Twitter search engine. Assuming an average filter size similar to this value and over 300 millions of users, the problem is how to efficiently determine the set $\Gamma^-(n)$ of followers based now on the graph 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.

We propose and compare 3 different index structures that extend the existing graph structure storage 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[7]), we consider structures that fit in memory. this discarded tree-based solutions (see for instance [7] for a hash-based vs tree based memory requirement comparison in an RSS context). 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. Our three 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 θ_{dir}, θ_{list} and θ_{entry} the size of respectively a directory, list and entry (follower's, publisher's or term's ids).

The set of parameters and notations that impact construction time, memory requirements and matching time are summarized in Table 1.

[7] http://yearinreview.twitter.com/en/tps.html

N	total number of accounts
$\Gamma^+(n)$	accounts followed by n
$\Gamma^-(n)$	accounts that follow n
φ	average number of followers for a user
τ	average number of filter terms for a (publisher,follower) pair
$\|p\|$	size (distinct terms) of a post p
(k, β)	Heaps' law coefficients for micro-blogging datasets
γ	Zipf's law constant for micro-blogging datasets
$\mathcal{V}_F$	filter vocabulary
θ_{dir},	size of a directory entry
θ_{list}	size of a posting list
θ_{entry}	size of an entry in a posting list

Table 1. Parameters that characterize micro-blogging systems

3.1 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 their filters: $Postings_{PFT}(n) = \{(n_1, \{t^1_{n_1}, t^2_{n_1}, \ldots\}), (n_2, \{t^1_{n_2}, t^2_{n_2}, \ldots\}), \ldots\}$, 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 of Section 1.

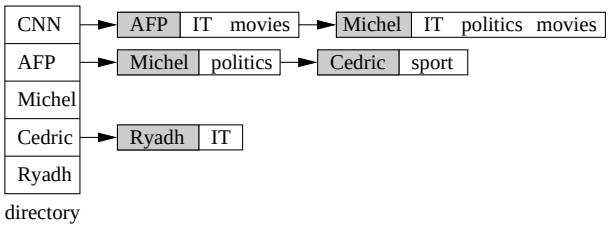

Fig. 2. The PFT-index

PFT-index memory requirement Let φ be the average number of followers for a user, and τ 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 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 $label(n_i)$ contains t_j. Thus the 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 exists 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 costs are respectively:

$$\begin{aligned} &\Delta^{PFT}_{insert}(FSG) = \varphi/2 \\ &\Delta^{PFT}_{delete}(FSG) = \varphi/2 + \tau/2 \end{aligned} \quad (3)$$

3.2 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 in Section 1.

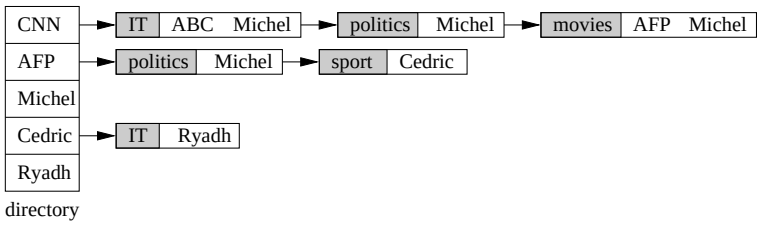

Fig. 3. The PTF-index

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* [2, 17], *i.e.*, $|Postings_{PTF}(n)| = k \times T^{\beta}$, where k and β are constants and T is the total number of terms in the posting. Heaps' coefficients k and β depend strongly on the characteristics of the analyzed text corpora and their value in our micro-blogging system has to be determined in future work. Note that β 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:

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

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 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 exists in $Postings_{PTF}(n)$, a new

entry for this term is added. Deleting requires an additional scan of the list of followers, on average $(\varphi \times \tau)/|Postings_{PTF}(n)|$. Potentially it leads to the deletion of a term's entry. Consequently the costs are respectively:

$$\Delta^{PTF}_{insert}(FSG) = k \times (\varphi \times \tau)^{\beta}/2 \quad (6)$$
$$\Delta^{PTF}_{delete}(FSG) = k \times (\varphi \times \tau)^{\beta}/2 + (\varphi \times \tau)/(k \times (\varphi \times \tau)^{\beta})$$

3.3 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 $\mathcal{V}_F$. The corresponding *posting list* $Postings_{TPF}(t)$ associated to a term t, is the set of publishers along 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 $n^j_i \in \Gamma^-(n_i)$ and $t \in label(n_i, n^j_i)$. TPF-index corresponds to a factorization first on the term from$\mathcal{V}_F$, 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 in Section 1.

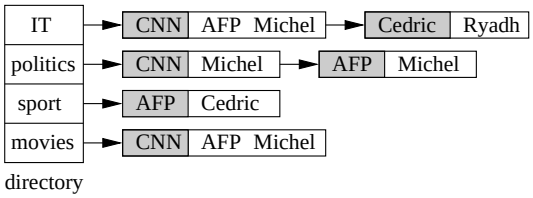

Fig. 4. The TPF-index

TPF-index memory requirement The TPF-index consists of $\mathcal{V}_F$ posting lists, one for each term of the filter vocabulary. Since we have N accounts that follow on average φ other accounts with an average filter size of τ, 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 $|\mathcal{V}_F| = k \times (N \times \varphi \times \tau)^{\beta}$. We make the common assumption that the distribution of terms in the set of filters follows the Zipf law [2, 17], 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 γ/r_i, where γ is a constant.

Here again $N \times \varphi \times \tau$ entries are finally stored, corresponding to the followers'ids. Thus the memory requirement for the TPF-index is:

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

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

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

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 $ln(|\mathcal{V}_F|)/2$, and we scan on average half of the posting to find the publisher. Thus the average matching time is:

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

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 $ln(|\mathcal{V}_F|)/2$ 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 exists in $Postings_{TPF}(t)$, a new entry is added. Deleting requires an additional scan of the list of followers, on average $(N \times \varphi \times \tau)/|\mathcal{V}_F|$. Potentially it leads to the deletion of a publisher's entry. Consequently the costs are respectively:

$$\begin{aligned}\Delta^{TPF}_{insert}(FSG) &= (\gamma \times ln(k \times (N \times \varphi \times \tau)^{\beta})/2)/2 \\ \Delta^{TPF}_{delete}(FSG) &= (\gamma \times ln(k \times (N \times \varphi \times \tau)^{\beta})/2)/2 + (N \times \varphi \times \tau)/|\mathcal{V}_F|)\end{aligned} \quad (9)$$

4 Experiments

In this section, we describe the experiments designed and conducted in order to analyze (i) how our structures behave against real micro-blogging data from *Twitter* and (ii) the impact of different parameters of micro-blogging systems. These experiments validate our analytical model presented in Section 3.

4.1 The dataset

In collaboration with Forth Institute (Heraklion, Crete)[8] we gathered data on Twitter over a four month-period using the Twitter streaming API[9]. We generated a complete Twitter *graph + tweets* dataset by merging this data with graph structures from other datasets available on the Web. We obtained an important 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 written ones (and their associated accounts). Our resulting dataset is summarized in Table 5.

Element	♯
Users	2,170,784
Tweets	15,717,449
Graph arcs	148,508,857

Fig. 5. Dataset description

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

Fig. 6. Filtered social graph dataset sample

To generate filters 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 [21]. We decide to generate filter terms for a follower among the most frequent and significative 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 "realistic" filter set is used for our experiments. Table 6 shows a sample of the filter social graph obtained using our generation process.

Experiments run on a Intel Core i5 CPU with a frequency of 3.60 GHz and 16 GB of RAM. The structures are implemented in JAVA.

4.2 Memory requirement

All structures have different factorization criteria which lead to different memory requirements. Table 2 compare 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,

[8] We thanks Vassilis Christophides for his support and helpful comments
[9] http://dev.twitter.com/

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 2. Index size for the realistic dataset

To measure the impact of the different parameters of our micro-blogging system and to validate our analytical model, we generate synthetic datasets with a constant number of filters (τ) for each graph edge. We report results in Figure 7. The memory occupancy grows linearly with τ for PFT and PTF indexes. Indeed, increasing τ 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 τ. 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 τ does not impact the structure but only the number of entries. Finally we note the low impact of τ on the TPF-index. The rationale is (i) the directory size remains constant and equal to $\mathcal{V}_F$, (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.

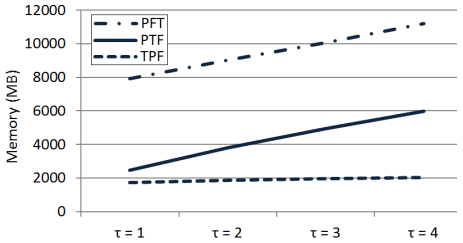

Fig. 7. Occupied memory w.r.t. τ

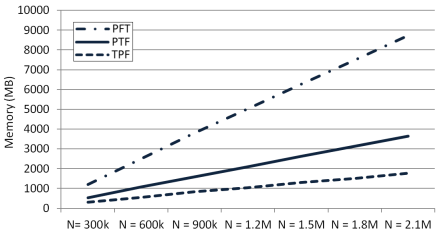

Fig. 8. Occupied memory w.r.t. N

Figure 8 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 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 (Equations 1, 4 and 7).

4.3 Indexing times

We compare in Table 3 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 τ. 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 (Equations 3, 6 and 9).

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

Table 3. Indexing times (in s)

4.4 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 4 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 15 μs, 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 4. Matching time for realistic dataset

Fig 9 illustrates the impact of τ on matching time. Like in Table 4 PTF-index outperforms other proposals with 2 orders of magnitude. We observe that the matching time for PFT-index linearly increases with τ 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 τ 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 τ, this explains this linear growth. For the TPF-index, we observe the Zipf's law behavior in the term frequency distribution, so when increasing τ 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. τ.

We report also in Fig 10 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 τ. These results also confirm our analytical model (Equations 2, 5 and 8).

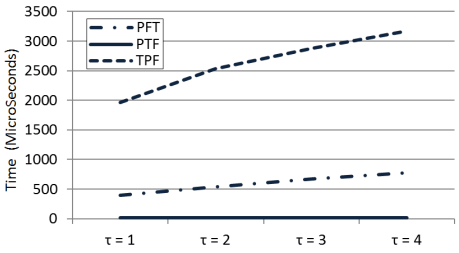

Fig. 9. Matching time w.r.t. τ

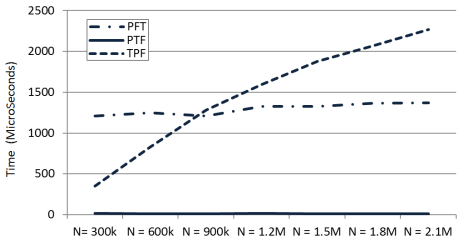

Fig. 10. Matching time w.r.t. N

4.5 Filtering efficiency

To evaluate how filtering reduces the number of posts delivered, we compare the number of posts that are sent from the micro-blog 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 5). 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 5. Number of delivered *posts*

5 Related Work

The recent success of *Twitter* and the important amount of user-generated content it handles have attracted the interest of the researchers' community making micro-blogging one of the trending research topics in the last few years. First studies attempt to capture the main characteristics of micro-blogging systems and analyzed the behavior of the users [9, 13, 14, 8, 23]. [9] presents one of the first studies that looked inside *Twitter*. It shows for instance that users with similar intentions connect with each other. [13] studied the following behavior and information diffusion patterns for a consequent snapshot of the entire Twittersphere. They also implement the *PageRank* algorithm on top of *Twitter* users. Other works focus on the lexical aspect of Twitter language and find similarities with

SMS language [14], [8]. [23] investigates the semantics of the twitter links, and finds that the retweet relation is a stronger indicator of the topical interest than the following behavior. We strongly rely on these works to propose structures that exploit micro-blogging characteristics, and to explain some experimental results.

Some papers improve data presentation to the users and attempt to avoid user flooding [20, 18, 3, 24, 22]. [20] presents a clustering and classification of tweets based on the users profile. It relies on a trained classifier to route an upcoming *tweet* to a predefined class. Classification can be also based on breaking news topics. In [18] for instance, the authors describe a technique for concept extraction from noisy source, and apply it to retrieve news from *Twitter* data. [24, 3] propose techniques to compute the user influence in *Twitter* and to rank tweets according to the user influence. [22] presents a filtering approach which takes advantage of the retweet behavior to bring more important tweets forward. All these approaches aim at avoiding the user flooding but do not diminish the number of delivered messaged by the centralized system. Oppositely our approach propose a scalable structure that handle the filtering process on the server's side reducing drastically the number of posts delivered.

More generally, the problem of delivering Web 2.0 data over an underlying graph has also been treated in several papers. [19] considers high frequency update feeds. Authors propose a method to selectively materialize user events over streams in order to handle the scalability of such systems (*Twitter's* timeline works that way[10]). [4] introduce the evaluation of graph constraints in content-based publish/subscribe systems. Authors suppose that the publishers and subscribers are connected by a directed graph (like in micro-blogging systems) and they implemented algorithms to efficiently evaluate constraints. Another filtering technique is presented by [6] where authors process top-k algorithms on top of Web 2.0 streams by considering the real-time aspect of such data. [7] proposes and compares indexing schemes for a pub/sub system that scales to millions of users and high publication rates. Our context is quite different since we have very short messages (posts) and extremely short queries (filters) but with a number of users and queries much more important.

6 Conclusion

In the present paper we compare three inverted lists-based structures that index filters to decrease the number of messages delivered in micro-blogging 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 memory requirements and insertion time twice more important than TPF-index, it outperformed with two orders of magnitude other proposals for matching time.

As future work we intend to improve the PTF-index in several ways. First we would like to exploit the heterogeneity of the accounts as reported in [13] (*e.g.*

[10] http://blog.evanweaver.com/2010/08/12/distributed-systems-primer-update/

5% of accounts with more than 100,000 followers). Clustering/Summarization is another option to group different filters inside a posting list to achieve better performance. Adding conjuction and negation in filters is another challenge. We also envisage to rely on ontologies to perform a more "clever" filtering (for instance considering synonymy or containment relationship). Finally with a more complex logic for filters expression and the use of semantics, one can consider ranking of tweets or at least a score function that determines whether the post is relevant for a user or not.

Acknoledgement: Michel Scholl passed away the 15^{th} of November 2011, too early. The authors would like to thank Michel Scholl for his devotion to the database research. We miss you.

References

1. Evan Weaver's Blog (Software Engineer at Twitter). http://blog.evanweaver.com.
2. R. A. Baeza-Yates and B. A. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press / Addison-Wesley, 1999.
3. E. Bakshy, J. M. Hofman, W. A. Mason, and D. J. Watts. Everyone's an Influencer: Quantifying Influence on Twitter. In *Proc. Intl. Conf. on Web Search and Web Data Mining (WSDM)*, pages 65–74, 2011.
4. A. Z. Broder, S. Das, M. Fontoura, B. Ghosh, V. Josifovski, J. Shanmugasundaram, and S. Vassilvitskii. Efficiently Evaluating Graph Constraints in Content-Based Publish/Subscribe. In *Proc. Intl. World Wide Web Conference (WWW)*, pages 497–506, 2011.
5. J. Foster, zlem etinolu, J. Wagner, J. L. Roux, S. Hogan, J. Nivre, D. Hogan, and J. van Genabith. #hardtoparse: POS Tagging and Parsing the Twitterverse. In *Proc. Intl. Work. on Analyzing Microtext (AMW)*, 2011.
6. P. Haghani, S. Michel, and K. Aberer. The Gist of Everything New: Personalized Top-k Processing over Web 2.0 Streams. In *Proc. Intl. Conf. on Information and Knowledge Management (CIKM)*, pages 489–498, 2010.
7. Z. Hmedeh, H. Kourdounakis, V. Christophides, C. du Mouza, M. Scholl, and N. Travers. Subscription Indexes for Web Syndication Systems. In *Proc. Intl. Conf. on Extending Database Technology (EDBT)*, pages 1–12, 2012.
8. Z. Hmedeh, N. Vouzoukidou, N. Travers, V. Christophides, C. du Mouza, and M. Scholl. Characterizing Web Syndication Behavior and Content. In *Proc. Intl. Conf. onWeb Information Systems Engineering (WISE)*, pages 1–12, 2011.
9. A. Java, X. Song, T. Finin, and B. L. Tseng. Why We Twitter: An Analysis of a Microblogging Community. In *Proc. Intl. Work. on Advances in Web Mining and Web Usage Analysis (SNA-KDD)*, pages 118–138, 2007.
10. F. Kivran-Swaine, P. Govindan, and M. Naaman. 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)*, pages 1101–1104, 2011.
11. A. C. König, K. W. Church, and M. Markov. A Data Structure for Sponsored Search. In *Proc. Intl. Conf. on Data Engineering (ICDE)*, pages 90–101, 2009.
12. H. Kwak, H. Chun, and S. B. Moon. Fragile Online Relationship: A First Look at Unfollow Dynamics in Twitter. In *Proc. Intl. Conf. on Human Factors in Computing Systems (CHI)*, pages 1091–1100, 2011.

13. H. Kwak, C. Lee, H. Park, and S. B. Moon. What Is Twitter, a Social Network or a News Media? In *Proc. Intl. World Wide Web Conference (WWW)*, pages 591–600, 2010.
14. G. Laboreiro, L. Sarmento, J. Teixeira, and E. Oliveira. Tokenizing Micro-blogging Messages Using a Text Classification Approach. In *Proc. Intl. Work. on Analytics for Noisy Unstructured Text Data (AND)*, pages 81–88, 2010.
15. R. Levering and M. Cutler. The portrait of a common html web page. In *ACM Symp. on Document Engineering*, pages 198–204, 2006.
16. S. Ma and Q. Zhang. A Study on Content and Management Style of Corporate Blogs. In *HCI (15)*, pages 116–123, 2007.
17. C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.
18. J. Sankaranarayanan, H. Samet, B. E. Teitler, M. D. Lieberman, and J. Sperling. TwitterStand: News in Tweets. In *Proc. Intl. Symp. on Geographic Information Systems (ACM-GIS)*, pages 42–51, 2009.
19. A. Silberstein, J. Terrace, B. F. Cooper, and R. Ramakrishnan. Feeding Frenzy: Selectively Materializing Users' Event Feeds. In *Proc. Intl. Conf. on Management of Data (SIGMOD)*, pages 831–842, 2010.
20. B. Sriram, D. Fuhry, E. Demir, H. Ferhatosmanoglu, and M. Demirbas. Short Text Classification in Twitter to Improve Information Filtering. In *Proc. Intl. Conf. on Research and Development in Information Retrieval (SIGIR)*, pages 841–842, 2010.
21. J. Teevan, D. Ramage, and M. R. Morris. #TwitterSearch: a Comparison of Microblog Search and Web Search. In *Proc. Intl. Conf. on Web Search and Web Data Mining (WSDM)*, pages 35–44, 2011.
22. I. Uysal and W. B. Croft. User Oriented Tweet Ranking: A Filtering Approach to Microblogs. In *Proc. Intl. Conf. on Information and Knowledge Management (CIKM)*, pages 2261–2264, 2011.
23. M. J. Welch, U. Schonfeld, D. He, and J. Cho. Topical Semantics of Twitter Links. In *Proc. Intl. Conf. on Web Search and Web Data Mining (WSDM)*, pages 327–336, 2011.
24. J. Weng, E.-P. Lim, J. Jiang, and Q. He. TwitterRank: Finding Topic-sensitive Influential Twitterers. In *Proc. Intl. Conf. on Web Search and Web Data Mining (WSDM)*, pages 261–270, 2010.
25. T. W. Yan and H. Garcia-Molina. Index Structures for Selective Dissemination of Information Under the Boolean Model. *ACM Transactions on Database Systems (TODS)*, 19(2):332–364, 1994.