An in-Browser Microblog Ranking Engine

Stéphane Frénot, Stéphane Grumbach

An in-Browser Microblog Ranking Engine

Stéphane Frénot[1] and Stéphane Grumbach[2]

[1] Université de Lyon
[2] INRIA

Abstract. Microblogs, although extremely peculiar pieces of data, constitute a very rich source of information, which has been widely exploited recently, thanks to the liberal access Twitter offers through its API. Nevertheless, computing relevant answers to general queries is still a very challenging task. We propose a new engine, the Twittering Machine, which evaluates SQL like queries on streams of tweets, using ranking techniques computed at query time. Our algorithm is real time, it produces streams of results which are refined progressively, adaptive, the queries continuously adapt to new trends, invasive, it interacts with Twitter by suggesting relevant users to follow, and query results to publish as tweets. Moreover it works in a decentralized environment, directly in the browser on the client side, making it easy to use, and server independent.

1 Introduction

The amounts of personal data accumulated at Internet scale constitute a resource, much like raw materials, with a huge economic potential. Google made the first demonstration of the power of these data in 2003 with its Flu Trend, which allows to monitor the flu activity in the world, based on the frequency in all languages of flu related search on its engine, and has been shown not only to be accurate but moreover ahead of disease control institutions [GMP+09].

Among the large Internet corporations handling users data, Twitter is the one that provides the most liberal access to them. Since its inception in 2006, Twitter grew at an exponential pace[3], with now over 100 million active users, producing about 250 million tweets daily. Although microblogs might seem restricted, Twitter has an amazing potential, it serves more than a billion queries per day, supports around a million Apps[4], and its projected advertising revenue of $250 million in 2012 is predicted to have doubled by 2014.

In this paper, we propose a new approach to handle social network data. Our objective is twofold. (i) First, develop **real time algorithms** to find the **most relevant data on any topic**, whether popular or not, by using any ranking techniques. (ii) Second, develop **continuous server-less solutions**, with algorithms running on the client side.

[3] `http://www.dazeinfo.com/2012/02/27/twitter-statistics-facts-and-figures-from-2006-2012-infographic/`

[4] `http://gigaom.com/2011/08/16/twitter-by-the-numbers-infographic/`

On Twitter, our first objective is to identify the most relevant tweets satisfying a query, the most relevant twitterers on a topic or the most important tags to listen to. Our notion of relevance is similar to a topic specific Pagerank, which could rely on any concept available in the social network system.

Extracting knowledge from tweets is a challenging task. Microblog data are rather unconventional. On one hand, they are like most web data, relying on both content and graph connections (followers, links to URL, etc.), but on the other hand, their content is rather peculiar, both for its form, extremely short, and for its substance, often very communication or notification oriented. Moreover, inputs as well as most generally outputs, are streams of data, which should be handled as most as possible in real time.

Our algorithms have the following characteristics. (i) The rankings are computed at query time, streams of results are produced in realtime, with an accuracy which increases progressively. (ii) Queries are autonomic since they are continuously refined by taking into accounts trends expressed in the output streams. (iii) The algorithms are associated with a twitterer, and interacts with Twitter, by publishing query results, and following relevant twitterers.

Our second objective is to develop systems which are server-less and work as soon as the client's side is active. This approach is very promising. First, functionalities which are not offered by Internet corporations, can be handled directly by the users on their CPU. More generally, this model of decentralized computation alleviates the burden of centralized systems, by pushing to the periphery computations that do not need to be centralized, as Facebook does with the Like button for instance. Complex computations on fragments of large data sets can be performed as well at the client side, as is done by systems such as seti@home.

We demonstrate our approach on Twitter, with the Twittering Machine, that we developed in javascript, and which runs directly in the browser of the client. It is associated with the personal account of the user, which will be used to follow relevant twitterers as well as publish query results, and works essentially as follows. The machine takes as input streams of tweets satisfying an initial query, which are immediately displayed and analysed. The most relevant twitterers are then computed, and it is suggested to follow them, adding to the input of the machine their tweets. The keywords in the query are modified according to query results. In fact, the query takes a stream of keywords as parameter. Query results are regularly suggested for publication.

The paper is organized as follows. In the next section, we present the Twittering Machine. Section 3 is devoted to the presentation of the plug-ins specifying the queries, while some experimental results are presented in Section 4.

2 The Twittering Machine

Twitter offers essentially two ways to access tweets, (i) by following specific already known twitterers, with their complete stream of tweets, or (ii) by using the search API. The latter provides a very efficient real time search [BGL+12], with a simple query language that allows to filter tweets on their content, as

well as on various attributes, such as their hashtags, URLs, locations, etc. It produces a small percentage of the theoretical result, but often sufficient to extract knowledge, and is widely used.

The main goal of the Twittering Machine is to evaluate complex queries over streams of tweets. It relies on computations local to the client side, and works autonomously, without any server except the queried social system, Twitter. This section presents its global functionality, illustrated in Fig. 1., and explains the implementation choices.

The machine takes as inputs incoming streams of tweets either results of search queries, or produced by followed twitterers, performs algebraic computation corresponding to stream queries, and produces as output, streams of tweets to display to the user, tweets to be produced by the user on a specific topic, as well as instructions for the management of followed twitterers. Output results are re-injected into the main system, thus leading to an autonomic querying system, which self-regulates.

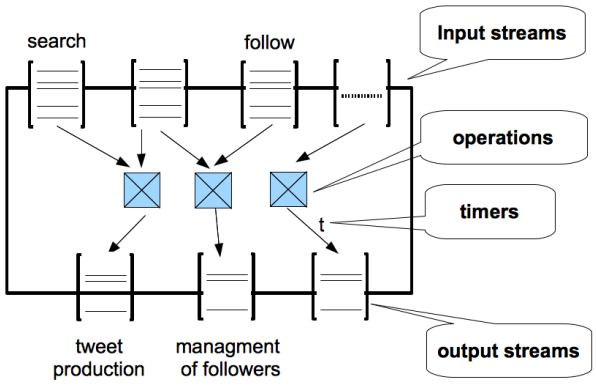

Fig. 1. The Twittering Machine

The Twittering Machine has the following main characteristics.

In-node execution We adopt a decentralized approach, with a system running mainly at the border of the network, on the end-user's computer, and to simplify its use, we choose to run directly at the browser level. With the rapid development of cloud architectures, more and more run-times are hidden at the clients side. Whenever possible, we always provide in-browser function execution. For cross-domain browser limitation reasons[5], we use a local run-time based on nodejs[6] that communicates locally with the browser and handles authentication requests. From the Twitter side, our in-node approach behaves as a standard end-user and develops a behavior similar to a real user.

Twitter friendliness The Twittering Machine interacts naturally with the API of Twitter. It uses all functionalities Twitter offers in its API. Twitter's data share

[5] `http://en.wikipedia.org/wiki/Same_origin_policy`
[6] `http://nodejs.org/`

with other web data, the duality of graph connections (followers) and contents (tweets), but its content is unconventional both in form and in substance. Tweets can be grabbed either from the general search API or from the followed twitterers through either REST or the stream based authenticated API. Followers can be managed with the same authenticated API as the REST API.

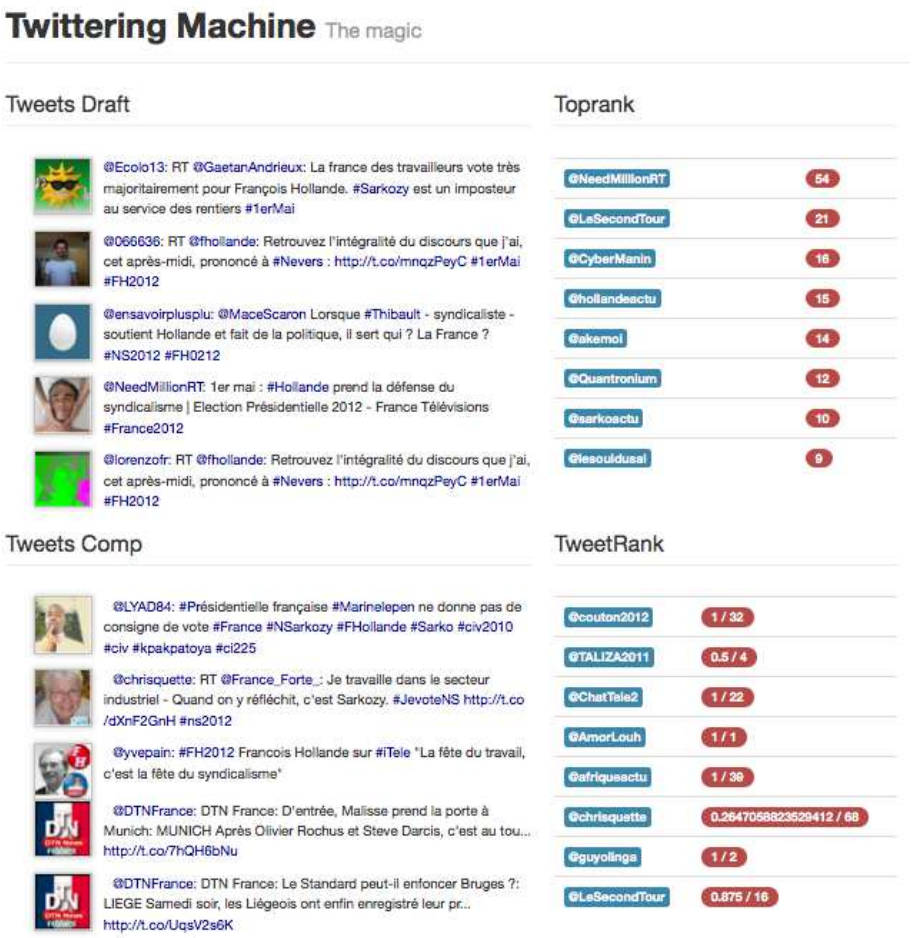

Fig. 2. The displayed tweets with their counts

Stream based plugin architecture The Twittering Machine is build around dynamic modules orchestrated into the browser. Each plugin works in a stream based way. It manages an input stream generally of tweets obtained through a dedicated query, realizes a grouping and an ordering of tweets and finally generates a stream for instance of twitterers as output. Plugins can be cascaded since the output from one plugin may be used as the input of another, as shown in Section 3. Finally, we consider every input and output parameters as streams. For instance the query parameters are extracted from an input stream and this in-

put stream can be adapted at run-time to include relevant keywords for instance that pop-up from the output computed so far.

Specifying plugins with a query Language Every plugin is expressed with a stream based query that uses the select-project-join-aggregate syntax of SQL as presented in Section 3. The ranking is defined using such queries which recursively lead to more accurate results.

Interaction with Twitter The Twittering Machine can suggest interactions with Twitter to the user, such as manage its follow relation, follow or unfollow twitterers, as well as query results of interest which can be twitted by the user. This interaction could be handled directly by the machine, but it would conflict with the Automation Rules and Best Practices from Twitter which prevents from automatic (un-)following. In our experiment, we have used a *Companion* account to interact with Twitter according to the recommendation of the machine.

Display of results The results are displayed in the user's browser. We illustrate the behavior of the machine on the query "Hollande", candidate to the French presidential election. On Fig. 2., the first series of tweets, under "Tweets Draft", satisfy a query extracted from Twitter, while the second series, under "Tweets Comp", are tweets produced by twitterers followed by the Companion.

The tweets visualized in Tweets Draft are ordered in decreasing order by twitterers that produced the largest number of tweets satisfying the query since the query was launched. The twitterer together with the number of corresponding tweets can be seen on the display. For the Tweets Comp, the twitterers are displayed together with the proportion of their tweets satisfying the query. Consider for instance the twitterer @LeSecondTour. It is associated with 21 tweets caught by the Tweets Draft, while for the *Companion*, 16 tweets (vs 21) have been caught (we started following this twitterer some time after the query was initially launched), and 87.5% of its tweets satisfy the query.

3 The Twittering Machine plugins

The Twittering Machine is based on plugins injected at run-time that enable the computation of queries. Each plugin is specified with a dedicated descriptive SQL-like specification and the Twittering Machine hosts and schedules the runtime of each plugin. The descriptive language handles streams manipulation. Each stream interaction is triggered with clocks whose parameters are specified within the query. The generic query structure has a form of the following type:

```
Every <X> seconds,
insert into <OutStreamName> values <[Val]>
    Every <Y> seconds,
    compute <[Val] = function()>,
    from <InStreamName>,
    where <Request>.
```

This listing shows the two independent loops of the query. The internal loop queries every <Y> seconds an <InStreamName> stream and maintains an internal array of results $[Val]$. Then, every <X> seconds, the external loop takes every $[Val]$ and injects them into the <OutStreamName> stream.

From this main structure, we can express a query that produces on the display, every five seconds, an array of the most active twitterers on French election candidate François Hollande.

```
Every 10s,
insert into Console values [twitterers]
  Every 5s,
  [twitterers] = topK (screenname),
  from SearchAPI,
  where tweet like '%Hollande%'
```

At runtime, when this request is plugged into the Twittering Machine, the scheduling is controlled, such that all streams may not be requested too often. The machine indexes every requested stream and collects timing constraints, if a stream is too much in demand, the twittering machine will lower timing values. Moreover, for performance reasons, if the internal memory for maintaining the array of twitterers is overloaded, the Twittering Machine will reduce its size. Finally, if the Twittering Machine can host the plugin, it compiles the query and generates a equivalent JavaScript code fragment (in the current implementation, the code fragment is generated by hand) that interacts with Twitter. For instance, without considering the topK computation part, the following code is generated for the internal loop.

```
my.run = function (from) {
 $.getJSON("http://search.twitter.com/search.json?callback=?",
      {'since_id':from,'q':'hollande','rpp':'40','lang':'fr'},
    function(tweets) {
      if (typeof(tweets.results) !== 'undefined') {
        if(tweets.results.length > 0){
          $.each(tweets.results, function() {
            my.tweetsCpt++;
            DICE.ee.emit('draft::addTweet', this);
          });
        }
      }
    }
 setTimeout(function()
                {DICE.draft.run(tweets.max_id_str);},10000);
}
```

The Twittering Machine identifies every input and output as streams. We distinguish between the following streams.

- **Non mutable streams.** Those streams can be both used as input stream such as the main search twitter stream or as output stream as the internal console. These streams are mostly either read-only or write-only from the Twittering Machine perspective.
- **Authenticated streams.** These streams identify user interactions with the external system. For instance the Twittering Machine plugin may express actions such as insert a new follower within a user stream. If the user validates this action the insertion is authenticated.

– **Internal Streams.** These streams mimic an external stream, but for performance, security or privacy reasons they are not available outside the Twittering Machine. For instance, we consider the 'where request' as a flow of query units that can be improved with specific tags. This 'request specification' stream is internally maintained and is not subject to external announcement.

The Twittering Machine manages resources associated to local computations. Each plugin is dynamically deployed within the machine. At deployment time, the framework checks compatibilities with the current installed plugins. As soon as a plugin is validated, the framework monitors its behavior and checks whether it is compliant with Twitter's rules, as well as the amounts of memory and CPU it requires.

The stream SQL like queries describing the plugins are compiled into control behavioral constraints. Each input and output stream pace is dynamically bounded and adapted to the current computer load. Each plugin internal memory is evaluated and adapted when possible to the currently available memory. Our current target implementation uses the web browser javascript interpreter as the Twittering Machine runtime. We designed a plugin based framework that enables dynamic loading and hosting of requests with respect to available resources.

4 Experiments

We have run experiments on tweets related to the French presidential election. As we discussed above, there are many systems giving useful trends on the popularity of candidates, which are now widely used together with traditional opinion polls. They generally extract knowledge from tweets which are essentially seen as raw data, and are often not included in the output. The Twittering Machine on the other hand identifies the relevant tweets and the relevant twitterers, and produces them as output.

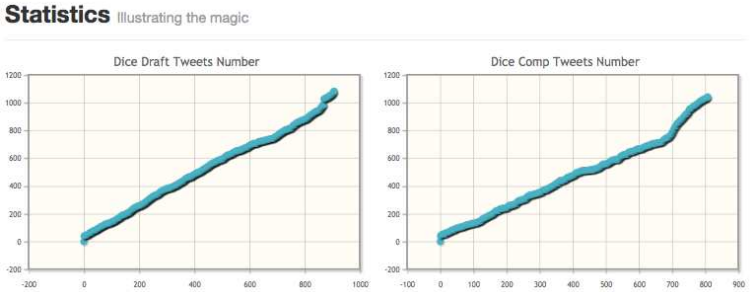

Fig. 3. Statistical information displayed

We illustrate its behavior on a simple query searching for the keyword "Hollande", the socialist candidate, whose result is shown on Fig. 2. The stream of

tweets obtained from Twitter's API is shown as Tweets Draft, while the stream of tweets obtained from followed twitterers is shown as Tweets Comp. The initial query was modified dynamically, with the evolution of the stream of keywords of the query, to which hashtags occurring in the most relevant tweets were added, in the limits permitted by the API.

Interestingly, the results obtained for other candidates overlapped massively with one another, for tweets are often comparative. We did not use any semantical or analytical methods, although they could clearly be combined to our approach, but it is out of the scope of this paper. The Twittering Machine produces real time statistics on the tweets satisfying the query, which are shown on Fig. 3. The curve on the left depends upon Twitter's search API, while the one on the right depends upon the twitterers followed. Interestingly, the two curves become very similar with time, showing the relevance of the choices of the machine.

Among the interesting results that this query allowed to grasp, was the hashtag #radioLondres, which was used by twitterers to publish results on the first round of the election before it was legally allowed in France.

5 Related work

There has been an exponential increase of academic work on techniques to extract knowledge from the data contained in tweets, while online systems to analyze them have flourished. The simplest systems offer graphical interfaces to watch the tweets themselves, their numbers, or relative numbers, and their dynamics. Semiocast[7] for instance produces a daily or monthly ranking of, for instance, major French politicians on Twitter with associated mood. It harvests the tweets continuously using queries on a set of keywords that can be updated, depending upon current affairs or evolving nicknames. It disambiguates them, and analyses their mood, and produces charts that are easily readable.

Twitter offers also various access to trends, which unlike Google trends, belong to the top trends only. They have been graphically organized on maps for instance with tools such as Trendsmap[8] which displays them on the world map. Apart from France, search for Sarkozy gave (on April 30) results in Jakarta, Kuala Lumpur and Brisbane, showing the sometimes awkward results of these tools. An equivalent of Flu Trend [AGL+11] has been developed based on similar principle as the one developed by Google. This illustrates the potential of Twitter whose tweets can be easily accessed, while the search queries on Google are out of reach of users and used exclusively by Google for commercial purposes.

Many systems rely on complex analysis of tweet data, including statistics, semantics, etc. This is the main focus of most of the papers of the workshop devoted to Making Sense of Microposts [RSD12]. Thompson [Tho12] defines a stochastic model to detect anomalies in streaming communication data that is applicable to information retrieval in data streams. Some systems also measure

[7] http://semiocast.com/barometre_politique
[8] http://trendsmap.com/

the influence, e.g. Klout[9], or the value in financial terms of a twitterer, e.g. Tweetvalue[10].

Our aim is to offer the capacity to find relevant tweets on any topic, satisfying any type of queries. Two issues are thus at stake, the query language and the ranking. Stream queries have attracted a lot of attention in the last ten years, before their use for social data became important, with SQL like languages [MWA+03], formalized in [GLdB07]. Markus et al. developed a streaming SQL-like interface, TweeQL, to the Twitter API [MBB+11a,MBB+11b]. The language in the Twittering Machine, differs essentially from TweeQL, for its richer interactions with Twitter. We also assume that the continuous queries need to evolve over time. This topic has been considered recently in [ESFT11].

Adaptive indexing mechanisms for tweets have been proposed distinguishing between most frequent query terms [CLOW11]. Ranking is as for other web data an important issue, which combines content information with graph connections, and requires a recursive computation. Mechanisms to rank tweets based on pagerank like techniques have been proposed[11]. Topic-sensitive ranking algorithm which take users interest into account [Hav03], have been considered for tweets as well [KF11]. Variants of ranking can be considered in the Twittering Machine, whose first version implements a simple algorithm computed at query time. Various notions of relevance have been introduced. Tao et al. [TAHH12] propose to combine topic independent aspects, such as hashtags, URLs, and authorities of the twitterers.

6 Conclusion

Extracting knowledge from tweets has attracted considerable interest thanks to the generous access Twitter gives through its API. Nevertheless, most tweets are related to notification substance, not always of immediate interest. Identifying relevant tweets and twitterers on a given topic, and not only for top trends is of great importance. In this paper we have presented the Twittering Machine which allows to get tweets according to a ranking, which is computed in a continuous manner by a query engine which runs a stream SQL like query language.

The Twittering Machine, coded in javascript, runs directly in the Browser. It requires essentially no server, runs on the client and relies directly on the API of Twitter. We believe that this server-less approach is extremely promising. We plan to extend the present machine to allow collaboration between users interested in similar queries. The objective is to maintain the computation of queries, by dynamic groups of users, that are organized in a peer to peer fashion, and contribute to the update of the output stream of the query when they consult it, much like a torrent. This *query torrent*, will rely on the cooperation of peers for the computation of queries, with therefore more computation involved than file torrents. This approach raises security issues related in particular to communication between browsers that we are investigating.

[9] `http://klout.com/home`

[10] `http://tweetvalue.com/`

[11] `http://thenoisychannel.com/2009/01/13/a-twitter-analog-to-pagerank/`

Acknowledgements The authors would like to thank Fabien Culpo, who participated in the implementation of the Twittering Machine.

References

[AGL+11] Harshavardhan Achrekar, Avinash Gandhe, Ross Lazarus, Ssu-Hsin Yu, and Benyuan Liu. Predicting flu trends using twitter data. In *IEEE Conference on Computer Communications Workshops (INFOCOM)*, 2011.

[BGL+12] Michael Busch, Krishna Gade, Brian Larson, Patrick Lok, Samuel Luckenbill, and Jimmy Lin. Earlybird: Real-time search at twitter. In *IEEE International Conference on Data Engineering (ICDE)*, 2012.

[CLOW11] Chun Chen, Feng Li, Beng Chin Ooi, and Sai Wu. Ti: an efficient indexing mechanism for real-time search on tweets. In *ACM SIGMOD International Conference on Management of Data, Athens*, 2011.

[ESFT11] Kyumars Sheykh Esmaili, Tahmineh Sanamrad, Peter M. Fischer, and Nesime Tatbul. Changing flights in mid-air: a model for safely modifying continuous queries. In *ACM SIGMOD International Conference on Management of Data, Athens*, 2011.

[GLdB07] Yuri Gurevich, Dirk Leinders, and Jan Van den Bussche. A theory of stream queries. In *11th International Symposium on Database Programming Languages, DBPL, Vienna*, 2007.

[GMP+09] J. Ginsberg, M. Mohebbi, R. Patel, L. Brammer, M. Smolinski, and L. Brilliant. Detecting influenza epidemics using search engine query data. *Nature*, 457:1012–1014, 2009.

[Hav03] Taher H. Haveliwala. Topic-sensitive pagerank: A context-sensitive ranking algorithm for web search. *IEEE Trans. Knowl. Data Eng.*, 15(4):784–796, 2003.

[KF11] Shoubin Kong and Ling Feng. A tweet-centric approach for topic-specific author ranking in micro-blog. In *7th International Conference on Advanced Data Mining and Applications, ADMA*, pages 138–151, 2011.

[MBB+11a] Adam Marcus, Michael S. Bernstein, Osama Badar, David R. Karger, Samuel Madden, and Robert C. Miller. Processing and visualizing the data in tweets. *SIGMOD Record*, 40(4):21–27, 2011.

[MBB+11b] Adam Marcus, Michael S. Bernstein, Osama Badar, David R. Karger, Samuel Madden, and Robert C. Miller. Tweets as data: demonstration of tweeql and twitinfo. In *ACM SIGMOD International Conference on Management of Data*, 2011.

[MWA+03] Rajeev Motwani, Jennifer Widom, Arvind Arasu, Brian Babcock, Shivnath Babu, Mayur Datar, Gurmeet Singh Manku, Chris Olston, Justin Rosenstein, and Rohit Varma. Query processing, approximation, and resource management in a data stream management system. In *CIDR*, 2003.

[RSD12] Matthew Rowe, Milan Stankovic, and Aba-Sah Dadzie, editors. *Proceedings, 2nd Workshop on Making Sense of Microposts (#MSM2012): Big things come in small packages, Lyon, France, 16 April 2012*, April 2012.

[TAHH12] Ke Tao, Fabian Abel, Claudia Hauff, and Geert-Jan Houben. What makes a tweet relevant for a topic? In Rowe et al. [RSD12], pages 49–56.

[Tho12] Brian Thompson. The early bird gets the buzz: detecting anomalies and emerging trends in information networks. In *Proceedings of the fifth ACM international conference on Web search and data mining*, WSDM '12, 2012.