

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO



FEUP

Scheduling methods for distributed Twitter crawling

Andrija Čajić

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Prof. Eduarda Mendes Rodrigues (Ph.D.)

Second Supervisor: Prof. dr. sc. Domagoj Jakobović (Ph.D.)

June 18, 2012

Scheduling methods for distributed Twitter crawling

Andrija Čajić

Mestrado Integrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

Chair: Prof. João Correia Lopes

External Examiner: Prof. Benedita Malheiro

Supervisor: Prof. Eduarda Mendes Rodrigues

June 18, 2012

Abstract

Online social networking is assuming an increasingly influential role in almost all aspects of human life. Preventing epidemics, decreasing earthquake casualties and overthrowing governments are just some of the exploits "chaperoned" by the Twitter online social network. We discuss advantages and drawbacks of using the Twitter's REST API services and their roles in the open source crawler TwitterEcho. Crawling the Twitter user profiles implies a real-time retrieval of fresh Twitter content and keeping tabs on changes in relations between users. Performing these tasks on a large Twitter population while preserving high coverage is an objective that requires scheduling of users for crawling. In this thesis, we describe algorithms that fulfill the presented objective using a simple technique of tracking users' activities. These algorithms are implemented and tested on the TwitterEcho crawler. Evaluation of the implemented scheduling algorithms show notably better results when compared to the scheduling algorithms used in the current release of the TwitterEcho crawler. We also provide interesting insights into activity patterns of Portuguese Twitter users.

Acknowledgements

During several months of my intense work on this thesis, I received a lot of help from my friends and co-workers. I would like to thank Arian Pasquali, Matko Bošnjak (GoTS), Jorge Texeira and Luis Sarmento for their contributions to this thesis.

Special thanks go to both of my supervisors Prof. dr. sc. Domagoj Jakobović for patience and administrative help and Prof. Eduarda Mendes Rodrigues for continuous support and advising.

Andrija Čajić

Contents

1	Introduction	1
1.1	Motivation and objectives	2
1.2	Thesis contributions	2
1.3	Structure of the Thesis	3
2	Literature review	5
2.1	Scheduling algorithms for Web crawling	5
2.2	Twitter crawling systems	7
2.3	Tracking users' activity in the OSN	7
2.4	Summary	8
3	TwitterEcho	11
3.1	Twitter	11
3.2	Twitter API	13
3.3	Twitter API restrictions	14
3.4	Architecture	15
3.4.1	Server	15
3.4.2	Client	16
3.5	Summary	17
4	Scheduling Algorithms	19
4.1	Scheduling problem	19
4.2	Initial approach	22
4.2.1	Lookup service	22
4.2.2	Links service	22
4.3	New scheduling algorithm	22
4.3.1	Lookup service	23
4.3.2	Links service	29
4.3.3	Parameters	31
4.4	Summary	33
5	Evaluation and results	35
5.1	Comparing scheduling algorithms	35
5.2	Testing <i>inertia</i> parameter	38
5.3	Experimenting with starting activity	45
5.4	Effects of <i>online_time</i> parameter	45
5.5	Efficiency in distributed environment	48
5.6	Other evaluations	50

CONTENTS

5.7	Summary	51
6	Conclusion	53
6.1	Accomplishments	53
6.2	Future work	54
A	Implementation details	55
A.1	Cooling activity values	55
A.2	Increasing activity values	58
A.3	Accumulating activity	59
A.4	Links pagination	60
	References	63

List of Figures

3.1	Example use of hashtag for a topic	12
3.2	Example of a reply and mention	12
3.3	TwitterEcho architecture	15
4.1	Coverage vs. crawl frequency	20
4.2	Successful vs. wasted crawls	21
4.3	User's activity values	24
4.4	Activity changes at 18:00 upon retrieving new tweet which was created at 15:45. The last Lookup was performed at 13:00.	26
4.5	Cummulative activity from 15:15 to 17:45	27
4.6	Activity vs. tweet frequency	29
5.1	Scheduler comparison	37
5.2	Activity representation of user base with 87 978 users	37
5.3	Scheduler's predictions vs. realisation	39
5.4	"Conversational" tweets	39
5.5	Experimenting with <i>inertia</i> parameter	41
5.6	Activity values of the user #1 – 1 day <i>inertia</i> period	42
5.7	Activity values of the user #2 – 1 day <i>inertia</i> period	42
5.8	Activity values of the user #3 – 1 day <i>inertia</i> period	43
5.9	Activity values of the user #1 – 7 day <i>inertia</i> period	43
5.10	Activity values of the user #2 – 7 day <i>inertia</i> period	44
5.11	Activity values of the user #3 – 7 day <i>inertia</i> period	44
5.12	Experimenting with starting activity values	46
5.13	Crawlers using alternative values for <i>online_time</i> parameter	47
5.14	Ratio between the users selected for crawling based on the "online criterion" and the actual number of tweets retrieved from those users for 6-minute-online-time scheduler	47
5.15	Ratio between the users selected for crawling based on the "online criterion" and the actual number of tweets retrieved from those users for 12-minute-online-time scheduler	48
5.16	Tweet collection rates for variable number of clients	50
A.1	The TwitterEcho's simplified database diagram	56
A.2	The TwitterEcho's simplified database diagram after the implementation of the new scheduling approach	57

LIST OF FIGURES

List of Tables

5.1	Confusion table for tweet collection of both new scheduling algorithm and the one included in the latest TwitterEcho version	38
5.2	Top active users registered by different algorithms	40

LIST OF TABLES

Abbreviations

API	Application Programming Interface
CSV	Comma Separated Values
HDFS	Hadoop Distributed File System
HTTP	HyperText Transfer Protocol
JSON	JavaScript Object Notation
OSN	Online Social Network
Perl	Practical Extraction and Reporting Language
PHP	Hypertext Preprocessor
REST	REpresentational State Transfer
SQL	Structured Query Language
TfW	Twitter for Websites
URL	Uniform Resource Locator

Chapter 1

Introduction

Social networking is a natural state of the human existence. People have tendencies of making connections with other people, talking, sharing knowledge and experiences, playing games, *etc.* This is probably the main reason why the humankind accomplished so much in such a short period of time.

In the last half of a decade, social behavioral patterns in modern society underwent dramatic transformations. Online social networks (OSN) like Facebook, Twitter, Orkut and Qzone have "taken over" the Internet and the human interactions became more and more virtualized. The physical barriers have been lifted and we are witnessing the age of the fastest information distribution speed in history.

Online social networking is a global phenomena that enables millions of people using the Internet to evolve from passive information consumers to active creators of new and original media content. Access to popular social networks has become an indicator of democracy and equality among all people, while in some occasions it has even been put in the context with the basic human rights [[Hir12](#)].

Without going too deep into the analysis of repercussions of these fundamental changes, we can observe that online social interactions have retained a lot of properties of the traditional interpersonal relations within groups of people. The big difference, however, is that online communication is centralized and recorded while the real world communication is mostly distributed and nonpersistent. All communication and knowledge sharing taking place in the OSN is aggregated as a property of several leading companies, some of which are mentioned earlier in this section. In attempt to analyze this information scientists gather the relevant data by *crawling* the OSN.

The term "crawling" originates from a "Web crawler" – a type of computer program that browses the World Wide Web in a methodical, automated manner or in an orderly fashion [[Wik12](#)]. Crawling of OSN is a similar procedure with the exception that browsing is focused exclusively on user profiles in the OSN, the content they post and their mutual connections. The crawling process becomes more efficient and simplified if the OSN offers its services via an Application

Programming Interface (API) like it is the case with Twitter. Twitter's services can be accessed programmatically and it is, therefore, very suitable for crawling.

In this thesis we focus on crawling the Twitter OSN.

1.1 Motivation and objectives

From advertising, political or scientific point of view, data accumulated in the OSN is extremely valuable. Crawling of the OSN continually tries to extract this data in order to monitor real-time happenings, analyze public opinion, find interest groups, *etc.* For purposes like these it is important for the collected data to be up-to-date with the actual content in the OSN.

Due to the imposed restrictions on the usage of the Twitter's API, the goal is to achieve the maximum gain with the limited amount of resources. Specifically, it is preferable to acquire as much data that is both relevant and recent. From this aspect, it is possible to discuss crawling optimizations. In this thesis we explore new ways for optimizing the crawling of the Twitter OSN by using scheduling algorithms. We also introduce such an algorithm.

The approach proposed suggests tracking the users' activity patterns on Twitter and adjusting the crawling schedule accordingly so that the chosen Twitter API services are utilized to the maximum extent possible.

The goal of this thesis is to provide scheduling methods that will improve the crawler's efficiency in two important segments of crawling:

1. maximizing the coverage of the content the targeted Twitter population is posting;
2. keeping an up-to-date picture of social relations of the users the crawler is focusing on.

1.2 Thesis contributions

Scheduling methods presented in the thesis are evaluated on the TwitterEcho crawler – a research platform developed at the Faculty of Engineering of the University of Porto in the scope of the REACTION project¹ and in collaboration with SAPO Labs² [BOM⁺12]. Improved schedulers for the two types of crawling services used by TwitterEcho crawler, *Links* and *Lookup*, have been implemented. Evaluation indicates that the scheduling approach described in the thesis delivers much better results than the approach used in the current release of TwitterEcho crawler.

The crawler's Links service was modified to cope with the changes made by Twitter on 31st October, 2011 to the API used for collecting users' followers and friends. Several modifications were performed in the TwitterEcho's server and client communication in order to reduce the number of calls a client makes to the server.

This thesis also provides some insight into tweeting activity patterns of Portuguese Twitter users.

¹<http://dmir.inesc-id.pt/project/Reaction>

²<http://labs.sapo.pt>

1.3 Structure of the Thesis

The thesis is organized into 6 chapters.

In Chapter 2 we make a review of the recent literature related to the problem of crawl scheduling of the OSN.

Chapter 3 introduces the TwitterEcho crawler as the platform on which the scheduling methods are being tested. Inside, a quick overview of Twitter is provided as well as some of the Twitter API services.

Chapter 4 describes the scheduling problem encountered in crawling of the Twitter OSN and lays out some of the approaches taken to solve it. In chapter 5 evaluation of the suggested methods and the accompanying results are presented.

Finally, in Chapter 6 a conclusion is provided with a review of everything that was accomplished during the research, implementation and the evaluation period of the scheduling algorithm. Also, we provide suggestions for the future work regarding the TwitterEcho crawling scheduler. These suggestions include ideas for adding functionalities that could potentially improve the crawler's efficiency.

Introduction

Chapter 2

Literature review

The following chapter reviews the recent work related to the scheduling algorithms for crawling the OSN. Although no work has been found on this exact topic, a combination of related topics provides some useful information. Collected literature can be roughly divided into three categories:

- scheduling algorithms for Web crawling;
- Twitter crawling systems;
- activity tracking of users in OSN.

We will discuss all of them in sections that follow.

2.1 Scheduling algorithms for Web crawling

Both of the following studies use complex mathematical abstractions for modeling the Web's unpredictable nature.

The article on "*Optimal Crawling Strategies for Web Search Engines*" by Wolf *et al.* [WSY⁺02] addresses several problems regarding efficient Web crawling. They propose a two-part scheme to optimize the crawling process. The goals are the minimization of the average level of staleness of all the indexed Web pages and minimization of *embarrassment level* – the frequency with which a client makes a search engine query and then clicks on a returned Uniform Resource Locator (URL) only to find that the result is incorrect.

The first part of the scheme determines the (nearly) optimal crawling frequencies, as well as the theoretically optimal times to crawl each Web page. It does so within an extremely general stochastic framework, one which supports a wide range of complex update patterns found in practice. It uses techniques from the probability theory and the theory of resource allocation problems which are highly computationally efficient.

The second part employs these crawling frequencies and ideal crawl times as input, and creates an optimal achievable schedule for the crawlers.

Pandey and Olston [PO05] studied how to schedule Web pages for selective (re)downloading into a search engine repository and how to compute the priorities efficiently. The scheduling objective was to maximize the quality of the user experience for those who query the search engine. They show that the benefit of re-downloading a page can be estimated fairly accurately from the measured improvement in repository quality due to past downloads of the same page.

Hurst and Maykov [HM09] outlined a scheduling approach to Web log crawling. They stated the requirements an effective Web log crawler should satisfy:

- low latency,
- high scalability,
- high data quality,
- appropriate network politeness.

Describing the challenges that arose when trying to accommodate these requirements they listed the following:

- Real-time – The information in blogs is time-sensitive. In most scenarios, it is very important to obtain and handle a blog post within some short time period after it was published, often minutes. By contrast, a regular Web crawler doesn't have this requirement. In the general Web crawling scenario, it is much more important to fetch a large collection of high-quality documents.
- Coverage – It is important to fetch the entire blogosphere. However, if resources do not allow this, it is more desirable to get all data from a limited set of blogs, rather than less data from a bigger set of blogs (these two aspects of coverage may be termed comprehension and completeness).
- Scale – The size of the blogosphere is on the order of few hundred millions blogs.
- Data Quality – The crawler should output good quality, uncorrupted data. There should be no Web spam in the output.

A scheduling subsystem was implemented to ensure that the resources are spent in the best possible way. The scheduler uses URL priorities to schedule the crawling of Weblogs. The priority of the URL has a temporal and a static part. The static part is provided by the list creation system. It reflects the blog quality and importance. The static priority can also be set by an operator. The temporal priority of a blog is the probability that a new post has been published on the blog.

2.2 Twitter crawling systems

A general characterization of Twitter was done by Krishnamurthy, Gill and Arlitt [KGA08]. They performed the crawling of Twitter with no focus on any specific communities. During three weeks from January 22nd to February 12th, 2008 67 527 users were obtained. For tweet collection they used Twitter API service called "statuses/public_timeline". This service returns the 20 most recent statuses from all non-protected users.

Kwak *et al.* [KLPM10] studied the topological characteristics and its power for information sharing. They crawled Twitter from 6th to 31st of July 2009 using 20 whitelisted machines, with a self-regulated limit of 10 000 tweets per hour. The search started in breadth with Perez Hilton, who at the given time had more than one million followers. Searches over the Search API were conducted in order to collect the most popular topics (4 262 of them) and respective tweets. Topic search was carried out for 7 days for each new topic that arose. In total 41.7 million users, 1.47 billion social relations and 106 million tweets were collected.

In the research done by Weng, Lim, Jiang and He [WLJH10] the aim was to find the influential users on Twitter. They obtained users' mutual connections using Twitter API and their tweets using pure Web crawling. Tweet analysis was performed retrospectively to analyze users' tweeting habits. The crawling is continuous and the results presented comprised data collected from March 2008 to April 2009.

Benevenuto, Magno, Rodrigues and Almeida [BMRA10] dealt with the problem of detecting spammers on Twitter. They used 58 whitelisted servers for collecting 55.9 million users, 1.96 billion of their mutual connections and a total of 1.75 billion tweets. Out of all users, nearly 8% of the accounts were private, so that only their friends could view their tweets. They ignored these users in their analysis. The link information was based on the final snapshot of the network topology at the time of crawling and it is unknown when the links were formed. Tracking the users' changes in social relations is only possible through continuous crawling of users over longer periods of time.

2.3 Tracking users' activity in the OSN

A study done by Guo *et al.* [GTC⁺09] provides insights into users' activity patterns on a blog system, a social bookmark sharing network, and a question answering social network. Among other things their analysis shows that:

1. users' posting behavior in these networks exhibits strong daily and weekly patterns;
2. the user posting behavior in these OSN follows stretched exponential distributions instead of power-law distributions, indicating the influence of a small number of core users cannot dominate the network.

An analytical foundation is also laid down for further understanding of various properties of these OSN.

"*Characterizing user behavior in OSN*" is the title of the research done by Benevenuto, Rodrigues, Cha and Almeida [BRCA09]. The study analyzes users' workloads in OSN over a 12-day period, summarizing HyperText Transfer Protocol (HTTP) sessions of 37 024 users who accessed four popular social networks: Orkut, MySpace, Hi5, and LinkedIn. A special intermediary application called "Social network aggregator" was used by all users as a common interface for all social networks stated. The data that was analyzed is called *clickstream* data which is actually all recorded HTTP traffic between users and the "Social network aggregator". The analysis of the clickstream data reveals key features of the social network workloads, such as how frequently people connect to social networks and for how long, as well as the types and sequences of activities that users conduct on these sites. Additionally, they crawled the social network topology of Orkut, so that they could analyze user interaction data in light of the social graph. Their data analysis suggests insights into how users interact with friends in Orkut, such as how frequently users visit their friends' or non-immediate friends' pages. In summary, their research demonstrates the power of using clickstream data in identifying patterns in social network workloads and social interactions. Their analysis shows that browsing, which cannot be inferred from crawling publicly available data, accounts for 92% of all user activities. Consequently, compared to using only crawled data, considering silent interactions like browsing friends' pages, increases the measured level of interaction among users.

In the research performed by Wu *et al.* [WHMW11] general Twitter population was crawled in hope to answer some of longstanding questions in media communications. They found a striking concentration of attention on Twitter, in that roughly 50% of URL consumed are generated by just 20 000 elite users, where the media produces the most information, but celebrities are the most followed. They used a Twitter "firehose" service - the complete stream of all tweets.

2.4 Summary

In this chapter we reviewed some of the studies related to the issues discussed later on in this thesis.

The works regarding Web crawling schedulers often dominantly discuss topics like Web pages' relevancy, availability, server's quality of service, *etc.* These problems make Web crawling more complex than crawling of OSN. The one aspect they have in common is that pages need to be crawled proportionately to the frequency at which they refresh their content. When crawling the OSN, users need to be checked proportionately to the frequency they put new content online.

Twitter crawling systems pointed out some methods for retrieving data from Twitter. Twitter "firehose" returns all public statuses. This is the ultimate data retrieval tool. Unfortunately, it is currently limited to only a few Twitter's partner organizations like Google, Microsoft and Yahoo. Recent developments indicate that Twitter is making this service available to the public but for a price that is ranging up to 360 000 USD per year [Kir10].

The whitelisted accounts used in [KLPM10] do not exist since February 2011 [Mel11].

Literature review

Based on the reviewed studies that did not use privileged services like firehoses or whitelisted accounts, we concluded that the best approach for collecting "fresh" users' tweets and their mutual connections free of charge is a combination of Twitter's Streaming API with several important Representational State Transfer (REST) API services: "users/lookup", "followers/ids", "friends/ids".

Studies about users' activity tracking show some insights about what should be the approach in scheduling algorithms for crawling the OSN. They encouraged the use of REST API "users/lookup" service for crawling dynamic variable-sized huge lists of Twitter users. These lists are free to be expanded and reduced at any time without affecting the crawling process.

The static and temporal priorities introduced in the study by Hurst and Maykov [[HM09](#)] share much resemblance to the scheduling approach to crawling taken in this thesis.

Literature review

Chapter 3

TwitterEcho

TwitterEcho¹ is a research platform that comprises a focused crawler for the twittosphere, which is characterized by a modular distributed architecture [BOM⁺12]. The crawler enables researchers to continuously collect data from particular user communities, while respecting Twitter API's imposed limits. Currently, this platform includes modules for crawling the Portuguese twittosphere. Additional modules can be easily integrated, thus enabling to change the focus to a different community or to perform a topic-focused crawl. The platform is being developed at the Faculty of Engineering of the University of Porto, in the scope of the REACTION project and in collaboration with SAPO Labs. The crawler is available open source, strictly for academic research purposes. TwitterEcho project was used during the parliamentary elections in Portugal in 2011 with a mission to collect as many relevant tweets possible about prime-minister candidates [BOM⁺12].

At the moment of writing, TwitterEcho is fully focused on covering European Football Championship in Poland and Ukraine 2012.

In this chapter, we introduce Twitter and describe the TwitterEcho crawler.

3.1 Twitter

Twitter is a microblogging service that enables users to send and read text-based posts of up to 140 characters, known as "tweets". It was launched in July 2006 by Jack Dorsey, and has over 140 million active users as of 2012. It has a reputation of being the world's fastest public medium. A lot of scientific work has been done related to the real-time collection of Twitter data. For example, a real time detection of earthquakes [SOM10] or detecting epidemics by analyzing Twitter messages [Cul10]. Twitter has also been cited as an important factor in the Arab Spring [Sal11, BE09, Hua11] and other political protests [Bar09].

Twitter users may subscribe to other users' tweets – this is known as following and subscribers are known as followers. As a social network, Twitter revolves around the principle of followers.

¹<http://labs.sapo.pt/twitterecho>

Users that follow each other are considered friends. Although users can choose to keep their tweets visible only to their followers, tweets are publicly visible by default. A lot of users prefer to keep their tweets public, whether it is because they wish to increase the reach of their messages, because of advertising capabilities or some other reasons. Users can group posts together by topic or type by using hashtags – words or phrases prefixed with a "#" sign. It was created organically by Twitter users as a way to categorize messages. Clicking on a hashtag in any message shows all other tweets in that category. Hashtags can occur anywhere in a tweet – at the beginning, in the middle, or in the end. Hashtags that become very popular are often referred to as the “Trending Topics”.

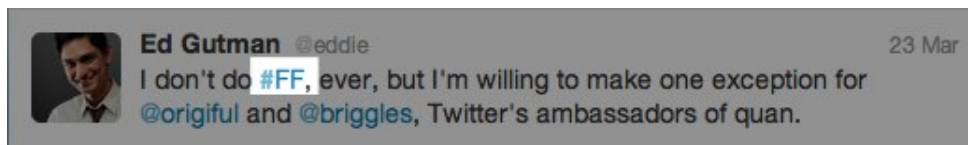


Figure 3.1: Example use of hashtag for a topic

In the figure 3.1, *eddie* included the hashtag *#FF*. Users created this as shorthand for "Follow Friday", a weekly tradition where users recommend people that others should follow on Twitter. Similarly to the use of hashtags, the "@" sign followed by a username is used for mentioning or replying to other users. A reply is any update posted by clicking the "Reply" button on a tweet. This kind of tweet will always begin with "@<username>". A mention is any tweet that contains "@<username>" anywhere in its body. This means that replies are also considered mentions. A couple of examples are shown in figure 3.2.



Figure 3.2: Example of a reply and mention

Twitter’s retweet feature helps users quickly share someone’s tweet with all of their followers. A retweet is a re-posting of someone else’s tweet. Sometimes people type "RT" at the beginning of a tweet to indicate that they are “re-tweeting” someone else’s content. This is not an official Twitter command or feature, but signifies that one is quoting another user’s tweet. Mentions and retweets are simple ways for users to expose their followers to something they consider interesting or even to serve as an intermediary for people to make new connections.

3.2 Twitter API

Web crawlers continuously download Web pages, index and parse them for content analysis. Twitter crawling is different in a way that the downloaded data does not need to be parsed in order to retrieve useful information. Instead, Twitter offers a lot of services through the API which deliver data in the JavaScript Object Notation (JSON) format. This reduces the network traffic load and speeds up the crawling process.

Each API represents a facet of Twitter and allows developers to build upon and extend their applications in new and creative ways. It is important to note that the Twitter API are constantly evolving, and developing on the Twitter platform is not a one-off event. Twitter API consists of several different groups of services [[Twi12](#)]:

- Twitter for Websites (TfW),
- Search API,
- REST API,
- Streaming API.

Each of them provides an access to a different aspect of Twitter.

Twitter for Websites (TfW) is a suite of products that enables Web sites to easily integrate Twitter. TfW is ideal for site developers looking to quickly and easily integrate very basic Twitter functions. This includes offerings like the Tweet or Follow buttons, which lets the users quickly respond to the content of a Web site, share it with their friends or get more involved in a newly discovered area.

The Search API designed for products and users that want to query Twitter content. This may include finding a set of tweets with specific keywords, finding tweets referencing a specific user, or finding tweets from a particular user.

The REST API enables developers to access some of the core primitives of Twitter including timelines, status updates, and user information. Through the REST API, the user can create and post tweets back to Twitter, reply to tweets, favorite certain tweets, retweet other tweets, and more.

The Streaming API allows for large quantities of keywords to be specified and tracked, retrieving geo-tagged tweets from a certain region, or have the public statuses of a set of users returned.

The TwitterEcho crawler uses three distinct methods for data extraction:

- Streaming – pure tweet collection;
- Lookup – gathering users’ tweets along with some other details about them;
- Links – discovering connections between users.

Streaming uses Twitter API service called “statuses/filter” which belongs to the category of streaming API. The set of streaming API offered by Twitter give developers low latency access to

Twitter's global stream of twitter data. This service is not based on request-response schema like the REST API services. Instead, the application makes a request for continuous monitoring of a specific list of Twitter users, a connection is opened and continuous streaming of new tweets is started. Each Twitter account may create only one standing connection to the public endpoints. An application will have the most versatility if it consumes both Streaming API and the REST API. For example, a mobile application which switches from a cellular network to WiFi may choose to transition between polling the REST API for unstable connections, and connecting to a Streaming API to improve performance.

"Statuses/filter" service allows a maximum of 5 000 user ids to be monitored per connection. Using exclusively streaming for data retrieval also implies that the current list of Twitter users that are being monitored is closed and is not going to change in foreseeable future. Some users involved in streaming may become irrelevant after some time therefore causing the inefficient utilization of streaming capabilities. It could happen because of their decreased tweeting activity, because they started being classified as bots or any other reason that requires them to be replaced with some other candidates.

Since streaming service monitors a fixed number of users, a self sustainable crawler needs an approach for monitoring a scalable list of Twitter users. The crawler currently relies on the Lookup service to crawl vast variable amount of users and to track their activity. The Lookup uses Twitter REST API service called "users/lookup". It returns up to 100 users worth of extended information, specified by either user's ID, screen name, or a combination of the two. The author's most recent status will be returned inline. This method is crucial for consumers of the Streaming API because it provides a platform of enormous user base from which streaming clients can pick users either based on their activity or any other criteria.

The TwitterEcho platform also includes the Links clients that crawl information about followers and friends of a given list of users. The Links uses a combination of REST API services "followers/ids" and "friends/ids" both of which return an array of numeric ID of all followers/friends of a specified user.

The main focus of this thesis is to propose techniques for optimizing the usage of the Lookup and the Links service. Those two services in combination with streaming service provide everything needed to have a stable and scalable crawler.

3.3 Twitter API restrictions

Twitter imposes restrictions on the usage of its API.

"Statuses/filter", the streaming service used, is limited to 5 000 users per connection while every Twitter account is limited to 350 REST API calls per hour.

"Users/lookup" spends 1 REST API call. In that call, information is being retrieved about a list of up to 100 Twitter users. Using a single client it is impossible to perform the Lookup service more than once per minute without decreasing the number of API calls used per hour. The crawler using a single client is, therefore, unable to collect multiple tweets posted within the same minute

by any given user, since the "users/lookup" API call only returns the last user's tweet. This is ultimately the biggest handicap of the Lookup service compared to the streaming services.

The "followers/ids" service, like "friends/ids", require 1 REST API call to collect maximum of 5 000 followers/friends of a specific user. For example, if a user has 13 000 followers 3 calls will be spent in order to get the complete list of that user's followers. Same applies to friends retrieval.

Thus, the Lookup call on one user costs 0.01 REST API call while the Links call costs a minimum of 2 API calls.

3.4 Architecture

Collecting a large amount of data requires a distributed system because of Twitter's limitations of API usage imposed on every Twitter account. The crawling platform includes a back-end server that manages a distributed crawling process and several thin clients that use the twitter API to collect data. The architecture of the crawler is described with the diagram in Figure 3.3.

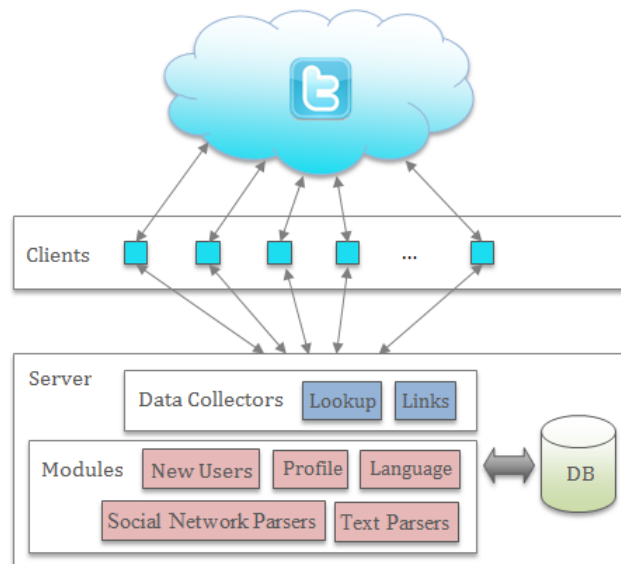


Figure 3.3: TwitterEcho architecture

3.4.1 Server

One of the main tasks of the server is to coordinate the crawling process by managing the list of users to be sent to clients upon request and maintaining the database of downloaded data. The modularity of the server enables user-control over both of these tasks. Regarding the crawling process, the server decides which users will be crawled and when.

In the current release of TwitterEcho, the Apache HTTP server is used as the server in the TwitterEcho architecture. All of the server side functionalities are implemented in Hypertext Pre-processor (PHP) and the MySQL relational database is used for data persistence.

3.4.1.1 Specialized modules

The server is initially configured with a seed user list to be crawled (*e.g.*, a list with a few hundred users known to belong to the community of interest) and continuously expands the user base using a particular expansion strategy. Such strategy is implemented through specific server modules, which need to be developed according to the research purpose. If the desired community is, *e.g.*, topic-driven, a new module would be implemented for detecting the presence to particular topics in the tweets. The corpus of users expands through a special module in two ways:

1. extracts screen names mentioned (@) or retweet (RT @) from the crawled tweets;
2. obtains the user ID from the lists of followers.

The server includes a couple of modules to filter users based on their nationality: profile and language. The current modules were specifically designed to identify Portuguese users, but they can be replaced and/or augmented by other filtering modules, *e.g.*, focused on other communities or focused on specific topics. The platform also includes modules for data processing – social network and text parsers that parse text posted in tweets and lists of followers and generate:

- network representations of explicit social networks (*i.e.*, network of followers and friends) and implicit social networks (*i.e.*, networks representing reply-to, mentions and retweets activities);
- network representations of #hashtags and URL usage patterns.

3.4.2 Client

The client is a lightweight, unobtrusive Practical Extraction and Reporting Language (Perl) script using any of three mentioned services: Streaming, Lookup or Links. Streaming and Lookup for collecting tweets, user profiles and simple statistics (*e.g.*, number of tweets, followers and friends count). The Links client script collects social network relations, *i.e.*, lists of friends and followers of a given set of users. Since these relations persist for longer periods of time there is no need to call the Links service on a particular user nearly as often as the Lookup service.

Clients using REST API (Lookup and Links) communicate to the server requesting and receiving "jobs". Jobs are lists of Twitter users server decided to crawl. After receiving a list, the client communicates with Twitter API requesting information about the users from the list. Upon collecting all necessary information, the client sends it back to the server that stores data into the database. The server includes a scheduler that continuously monitors the level of activity of the users and prioritizes the crawling of their tweets based on that level. Thus, the more active users are the more frequently their tweets get crawled.

Both scripts ensure continuity of the crawling process within the rate limits, thus respecting Twitter's access restrictions. It is also important to mention that one can easily increase the frequency of the crawling process by increasing the number of clients, assuming there is an adequate performance on the server side.

3.5 Summary

In this chapter we introduced the TwitterEcho research platform that contains a crawler which will be used in this thesis for testing the crawl scheduling algorithms. Twitter was quickly presented as the OSN targeted for crawling. We described some of its key concepts, its API and imposed restrictions. An introduction to the TwitterEcho's distributed architecture was provided and we discussed about client/server roles in the crawling process.

TwitterEcho

Chapter 4

Scheduling Algorithms

The following sections include the presentation of the scheduling problem for which this thesis aims to provide an adequate solution.

We analyze the TwitterEcho's current procedure for dealing with the scheduling problem and introduce a novel approach which was designed based on the experiences acquired during several months of using the initial scheduling algorithm.

Some ideas are also provided for anticipating and discovering the changes in users' mutual connections on Twitter.

4.1 Scheduling problem

After a preliminary analysis of data collected from the Portuguese twittosphere, it was observed that about 2.2% of users posted about 37% of the content, which highlighted the need to monitor active users' tweets more frequently than the inactive ones' in order to maximize the gain from a limited amount of the Twitter API calls. This prevents tweet loss for the most active users and ensures scalability of the system.

In order to achieve a self sustainable crawler, capable of expanding and reducing its own user base, REST API services need to be used with maximum efficiency.

A common phrase used in the context of evaluating crawler's efficiency in terms of maximizing collected tweets is *coverage*. Coverage is described as the amount of tweets collected by the crawler *vs.* actual number of tweets posted by the user. If all users were crawled equally frequent, low active users would get a 100% coverage while highly active users would be covered very poorly. On the other hand, if low active users are not to be crawled at all and all resources are spent on highly active users, tweet loss would be drastically reduced at the expense of ignoring users who tweet much less. If such users suddenly become very active, they would be completely overlooked. Thus, it is necessary to monitor all the users because some active users may become

inactive over time and vice-versa. Also, it is impossible to identify a list of users guaranteed to stay active for an unlimited period of time.

Using the Lookup service, for most of the users it is not possible to achieve 100% coverage. In fact, the relationship between crawl frequency and user coverage is nicely depicted in Figure 4.1. This happens because the Lookup service returns only the last tweets posted by a specified list of users.

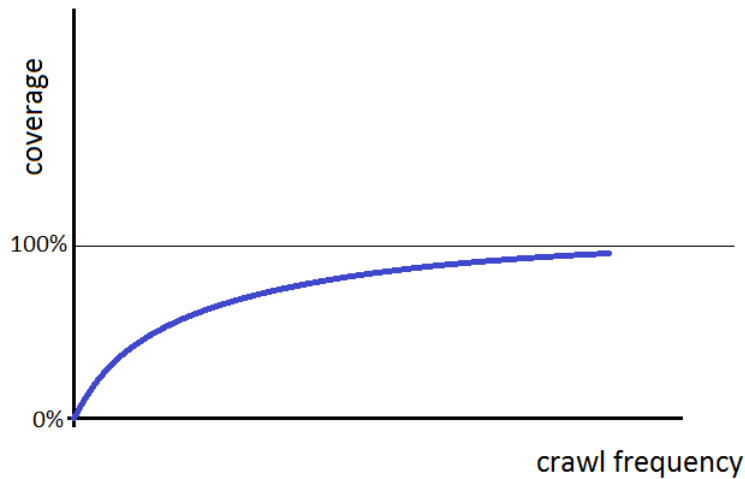


Figure 4.1: Coverage vs. crawl frequency

Each time a Lookup service is performed on the user, if a new tweet is not found, the crawl of that user is considered unsuccessful or wasted because it would have been better if the user was not included in that list of users for Lookup. Likewise, if a crawl of a specific user returns a new value (a new tweet), that crawl is considered successful. It is important to remember that each Lookup call to the Twitter API contains a list of 100 users to query. So, one request to the Twitter API consists of 100 crawls each of which can prove successful and justified or unsuccessful and unjustified.

If we display crawl frequency as sum of successful and wasted calls it would be like illustrated in Figure 4.2.

The goal can be considered as minimization of wasted API calls with the constraint that all, or at least a certain maximum number of available twitter API calls need to be utilized. Trying to maintain similar coverage of all users could be considered a *fairness constraint*.

While for the Lookup service scheduling purpose is to maximize the tweet collection rate, the Links service is slightly different. When mapping the connections between users, the goal is to have the social network charted in a graph form as precisely as possible at all times.

"Who follows whom?" is the basic Links question. Information gathered by the Links service is used afterwards not only by the crawler for user base expansion but also by other modules implemented in the TwitterEcho platform. Examples of such modules are: identifying influential users on Twitter, determining user's nationality or whereabouts, tracking tweet's origin, *etc.*

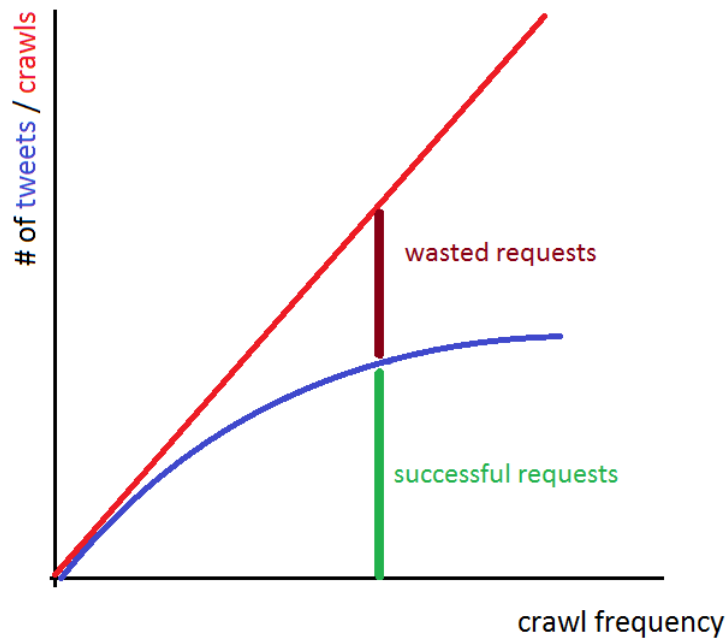


Figure 4.2: Successful vs. wasted crawls

Connections between users do not change very often. In Section 3.4.2 we already pointed out that the Links service does not need to be called as often as the Lookup service for one user. Neither it needs to be called at so precise moments in time. The reason why scheduling for the Links service is needed after all is that this service is much more "expensive" than the Lookup service (see Section 3.3). As a time consumption comparison, if a single designated client would do only the Lookup service around the clock, without the Links service, it would be possible to perform Lookup on 100 000 users in under **3 h**. In another situation where a client would do exclusively Links service all the time it would take more than **20 days** to check friends and followers of all 100 000 users. This fact shows that the prioritization of users for the Links service is as important as it is for the Lookup service, and maybe even more. The only difference is that changes in users' social connections do not happen nearly as frequently as posting tweets. Consequently, it would take weeks, if not months, for any scheduling algorithm to show its true efficiency.

Due to the crawler's distributed architecture, the scheduling algorithm must also be as scalable as possible, *i.e.*, the crawler should function well with variable number of clients, utilizing all of them to a highest possible extent. No matter how scalable the crawler is, on some level there will always be a limit to how many clients can work with the same server simultaneously. TwitterEcho is currently in the process of pushing those limits higher by transitioning to the HBase for data storage which is built on top of Hadoop Distributed File System (HDFS). This transition will make the data storage and retrieval faster and more consistent.

Determining the ratio in which the Lookup service and the Links service will be represented is also an implicit subproblem worth mentioning.

4.2 Initial approach

The initial approach to scheduling the crawling of users employs a simple heuristic for differentiating users based on their previously observed activity. A priority value is stored for each user for the Lookup service and another value for the Links service. The priority value is an integer ranging from 1 to 100.

4.2.1 Lookup service

Each time a Lookup call returns a tweet posted within the last h_i hours, the priority value increases by x . Likewise, each time a call confirms a lack of new tweets for the last h_d hours, the priority value decreases by y . Priority values help to decide which users to crawl and when. Based on those values, users are divided into five levels of activity. When assembling a group of users to be checked, each class nominates different number of users for Lookup or Links. The top activity level participates with highest number of candidates, and lower levels participate with less candidates. The changes in priority values for users allow them to move to upper level if an increase in activity is observed, and drop down to the lower level if a period of inactivity appears.

4.2.2 Links service

The Links activity is estimated based on comparing Comma Separated Values (CSV) strings containing lists of followers and friends of a queried user. Every time a newly retrieved string differs from the string that was last collected for that user, the Links priority value rises by a fixed amount. If a newly collected list of followers/friends is identical to the list from the last Links check-up on that user, the user's priority value is decreased by a fixed amount. Based on priority values, users are classified in 3 levels of Links activity and each level is assigned a different amount of resources, like with the Lookup service. Resources in this case are, of course, Twitter API calls.

4.3 New scheduling algorithm

The scheduling system implemented in the current release of TwitterEcho has lead to some problems that were not initially anticipated. The main flaws of the system mentioned can be described as follows.

Firstly, the scheduler relies on many user-defined parameters, chosen by empirical testing. If the scheduler starts performing badly, manual adjustment of parameters is required in order to achieve better performance for which it is again unsure how close it is to the optimum. Manual adjusting is always an estimation of a person who has some expertise in this area.

Secondly, considering the Lookup scheduling, it classifies users in 5 levels of activity. Users of the same activity level are crawled equally frequent. Division by 5 levels is a bit rough and is not expressive enough to fully achieve the goal – to crawl each user in direct proportion to their tweeting frequency.

Thirdly, it is not theoretically grounded. Activity levels are created based on users' priority values. So, a situation can occur when there is 5% of all users in the highest activity level, and another situation can occur when there is 15% of users in the highest activity level. That means there is no clear real-world interpretation of what the priority value is, or what do 5 levels of activity represent (other than members with higher priority are more active than those of lower priority).

Because of the shifty foundations that it stands on, attempts made to improve the scheduler's efficiency were performed by adding special rules and exceptions to the original idea. Here are some examples of such rules:

- users' priority values are not to be decreased during night (which is from 23:00 to 09:00) because assumption is made that almost all users are inactive during the night;
- user may not be crawled more than once in 20 s;
- crawling during the night is performed only on those users that are considered inactive.

In this thesis we propose a new scheduling algorithm that is designed to be more simple and robust in concept and more efficient overall.

4.3.1 Lookup service

We start with the assumption that users tend to exhibit a certain tweeting pattern throughout a day. For example, they might post new tweets in the morning, during lunch break at work, after work, before sleep, *etc.* This assumption is not the precondition to the scheduler's functionality. It only means that the scheduler is built with the ability to exploit such behavioral patterns if they do exist. To cite an instance, Figure 4.3 shows the tweeting activity of a user by hour of the day, which indicates a period of low activity during the night and high activity in the late afternoon. It is this type of patterns we aim to capture.

By acknowledging these patterns, optimization is achievable if users could be crawled at the time when they are most active. This kind of activity tracking is implemented in a way that instead of one priority value per user, the scheduler keeps a record of 24 activity values, one for each hour in a day (Figure 4.3).

Each time a new tweet is acquired, activity values around the time in a day when a tweet was created increase with total amount of 1.

However, activity values formed like this are still of no practical use. Firstly, because they are constantly rising, thus creating inequalities between users that have been monitored for a long time and the ones that are relatively new on the crawling list.

The second reason is that the user's activity is recorded over infinite amount of time. This is not sensible because when trying to predict the user's activity, the number of tweets per day from a year ago has little or no correlation with user's tweeting activity today. Users may not keep their tweeting patterns over extended periods of time. From time to time they abandon old and adopt new patterns. That is why the importance of user's recorded activity should fade with time. In

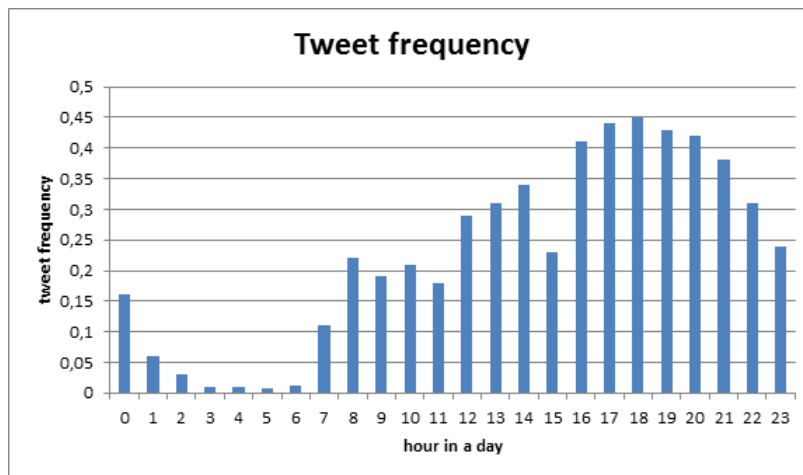


Figure 4.3: User's activity values

other words, the activity recorded in this week is much more important than the activity recorded a month ago if a goal is to predict when a user will tweet next.

Taking this into account, the new scheduler should keep the activity values shrinking constantly over time at a small rate. An analogy can be made between user's activity points and hot air balloons. While user is inactive they are constantly cooling off and dropping but when activity is noticed, it adds heat and lifts them up a bit. So the idea is actually that activity values of every user come to a point of equilibrium where during an arbitrary time interval the amount of activity that is cooled off is the same as the amount of activity added. In other words, every balloon will eventually find its maintainable altitude. Only then it can be said that the user is not crawled nor too often nor too rarely... This is how activity tracking for the Lookup service with cooling looks like – each time a Lookup has been performed on a user the following procedure is executed.

1. The user's activity values cool off based on how much time passed since the user was last checked.
2. The designated increment in activity points cools off based on how much time passed since the creation of the new tweet from the user.
3. Specific activity values are boosted around the time when the user's new tweet was posted.

4.3.1.1 Cooling

Activity cooling should decrease the user's activity values to the extent proportionate to a duration of time elapsed since the last time the user was crawled. Since the cooling is performed when a user is being crawled and only then, activity values are being cooled based on the time passed since the last time they were cooled.

The initial idea for cooling activity points is stated in Expression 4.1.

$$activity \leftarrow activity \cdot \frac{time}{time + inertia} \quad (4.1)$$

This approach satisfies the basic requirement in a way that it shrinks the activity value proportionately to the amount of time passed since the last crawl. The *inertia* parameter affects how fast will the cooling be. On the other hand, for example, if a users was crawled twice in the same hour, activity will be shrunk more than if a crawl was done only once during that hour. This indicates that the approach mentioned does not have fixed cooling speed for all users.

$$activity \leftarrow activity \cdot 0.5^{\frac{-time}{inertia}} \quad (4.2)$$

The approach defined with Expression 4.2 is exponentially decreasing activity over time. It has the same first basic property as the previous but it does not make distinction in cooling speed based on how often a user is crawled. The *inertia* parameter in this case could be interpreted as the time of inactivity required for the activity value to be cut in half. This is similar to what is called in chemistry the "time of half-life".

$$activity \leftarrow activity \cdot e^{\frac{-time}{inertia}} \quad (4.3)$$

The last proposed cooling mechanism (Expression 4.3) has all the properties mentioned in the previous approaches but has an extra quality which makes the activity points interpretable. This will be explained later on, in Section 4.3.1.5. As a part of standard Lookup of a particular user, cooling is performed every time.

The implementation of the described cooling approach is presented in appendix A.1.

4.3.1.2 Increasing activity

Unlike cooling, increase of activity does not happen for every user each time a Lookup is performed. Activity values are increased only if the Lookup call returned a new tweet (one that is not yet stored in the database). If a new tweet was collected the moment after it was created, then the amounts added to activity values sum up to 1. On the other hand, if a new, unregistered tweet is retrieved but that tweet is old by itself (some time has passed since its creation) then the increment values that are supposed to be added to activity values are also cooled based on how much time elapsed from tweet's creation to its retrieval. In other words, activity values always behave like all the tweets were collected the second after they were created. One example of activity changes after Lookup service is shown in Figure 4.4.

In appendix A.2 we outline the exact procedure for increasing the activity values.

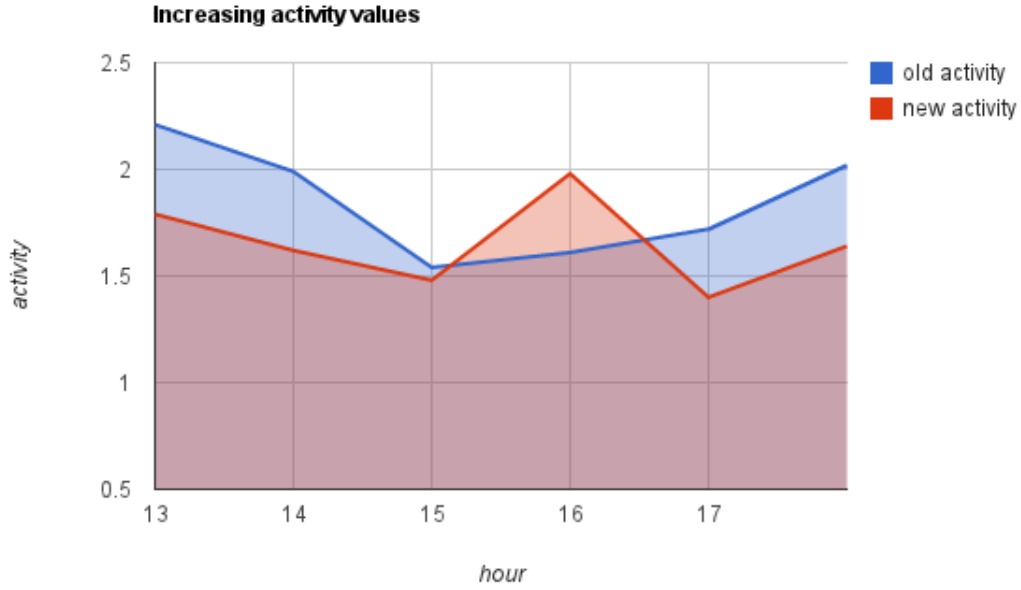


Figure 4.4: Activity changes at 18:00 upon retrieving new tweet which was created at 15:45. The last Lookup was performed at 13:00.

4.3.1.3 Accumulated activity

Activity values help the server to decide which users should be crawled and when. This is done via *accumulated activity*. Accumulated activity are summed up user's activity values since the last time that user was crawled. So at any given point in time, each user has some accumulated activity. For more active users, activity accumulates faster and for all users activity accumulates faster than usual around the time of day at which they are most active. The cumulative activity is illustrated in Figure 4.5.

The server's job of picking the users for crawling follows a trivial rule. The server sorts all users by their accumulated activity and picks the top users for Lookup. After they are picked for Lookup, their accumulated activity is annulled (set to 0). So, none of the immediate subsequent request from other clients (which can come within a few seconds after) will pick the same user for Lookup unless the user is extremely active. But activity starts accumulating again and based on how active the users are, they will get a chance sooner or later to be re-crawled.

Details on the implementation of activity accumulation are provided in appendix A.3.

4.3.1.4 Crawling online users

Another, independent aspect of scheduling is based on the fact that the period between two tweets posted by the average user varies quite a lot. This happens because users have online and offline periods. During the users' online time, they can post several tweets within just a couple of minutes.

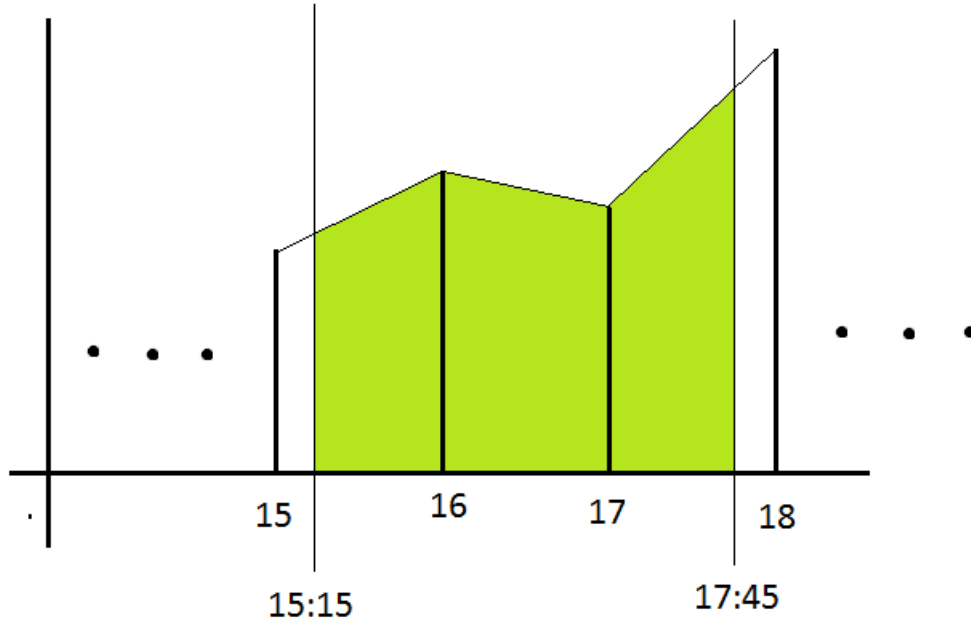


Figure 4.5: Cumulative activity from 15:15 to 17:45

After that user logs off and much longer period of inactivity starts. After a tweet was retrieved from a user, the information is also available about when the tweet was created. If a scheduler notices that a tweet was created within last *few minutes*, it can assume that this user might still be online. This user will therefore automatically be included in the next Lookup. The scheduler uses a special parameter called *online_time* which indicates what do those *few minutes* mean exactly for a given user. Tweets acquired solely based on crawling the online users are referred to as "conversational" tweets throughout the rest of the thesis.

4.3.1.5 Theoretical grounding

Described system of collecting tweets and "keeping score" makes the users' activity values constantly converging to the values that best describe their recent tweeting activities.

Due to the fact that exponential cooling mechanism is being used with the base of a natural logarithm e , the cooling speed of activity values is the same for all users no matter how often they are crawled. Therefore, for cooling purposes, it is irrelevant how many times a user was crawled unsuccessfully. The only thing that actually makes the difference in the activity values is the number of collected tweets per unit of time.

For example, in the hypothetical situation the user *A* and the user *B* have the identical activity points. After some time the user *A* was crawled 3 times and the user *B* was crawled 10 times. The user *A*, on the one hand, delivered a new tweet for each of the 3 crawls. On the other hand, the user *B* delivered a new tweet also 3 times but out of 10 crawls performed. If the times of occurrences of the tweets

Scheduling Algorithms

were identical for both users they would have identical activity points after the observed period passed.

The model is shown in Expression 4.4 that describes how the sum of user's activity values changes with the retrieval of a new tweet from that user.

$$A \leftarrow A \cdot e^{\frac{-t}{I}} + 1, \text{ where} \quad (4.4)$$

A = activity

t = time elapsed since the last tweet [s]

I = inertia parameter [s]

e = base of a natural logarithm

This change of activity values will either increase a total activity sum if it was previously too small, or decrease it if it was previously too large. It will only achieve balance if the new value is identical to the previous one. This situation is shown in Expression 4.5 with addition that variable t in this case represents an average time interval between tweets expressed in seconds.

$$\begin{aligned} A \cdot e^{\frac{-t}{I}} + 1 &= A \\ A \cdot (1 - e^{\frac{-t}{I}}) &= 1 \\ A &= \frac{1}{1 - e^{\frac{-t}{I}}} \end{aligned} \quad (4.5)$$

At this point, the new variable f is introduced.

$$\begin{aligned} f &= \frac{I}{t} \text{ (average number of tweets per } \textit{inertia} \text{ time)} \\ A &= \frac{1}{1 - e^{\frac{-1}{f}}} \end{aligned} \quad (4.6)$$

If f is high enough (this can be accomplished by tweaking the *inertia* parameter) the equation 4.6 is satisfied only when $f \approx A$ which is exactly what the activity value A should be – a reflection of a user's tweet frequency. Figure 4.6 plots the relationship between activity and tweet frequency stated in Expression 4.6.

$$\lim_{f \rightarrow \infty} \frac{1}{1 - e^{\frac{-1}{f}}} = f \quad (4.7)$$

Expression 4.7 shows that the higher the tweet frequency is, the more precisely the activity

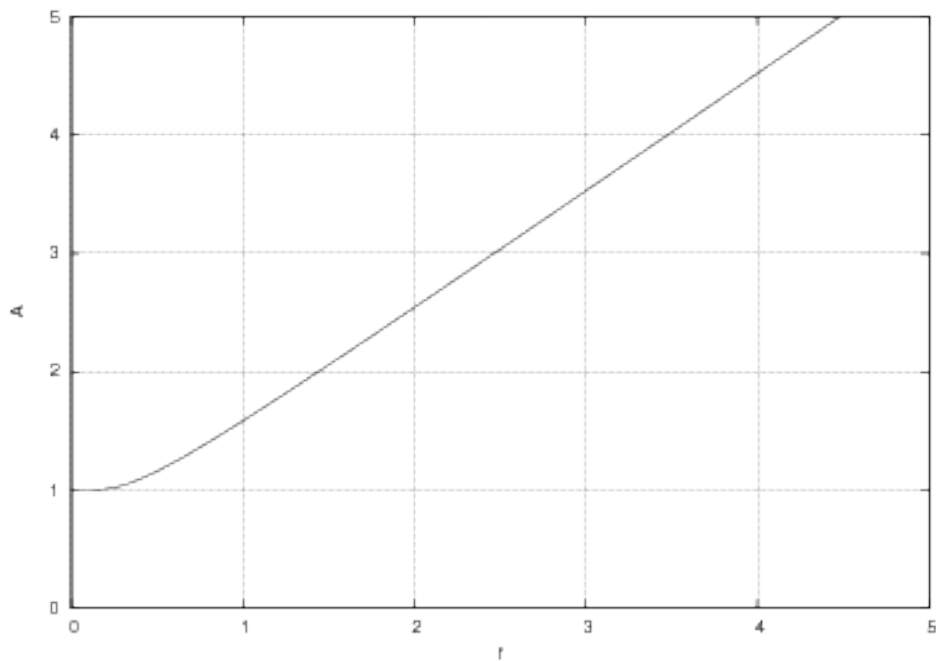


Figure 4.6: Activity vs. tweet frequency

values will represent it. Tweet frequency, in this case, is the number of tweets per time specified by the *inertia* parameter.

4.3.2 Links service

The need for a quality Links scheduling is already mentioned in Chapter 4.1. The goal is to determine which users are likely to have most severe changes in their followers or friends lists. There are no obvious patterns in time regarding acquiring or losing followers or friends.

Several criteria have been implemented in hope that some combination of them will give satisfying results. Criteria for choosing the users for the Links crawling:

- Links activity
- Tweet activity
- Account lifecycle
- Mention and retweet activity

Due to the changes in the Twitter API that occurred since the release of the current version of the TwitterEcho crawler, we implemented a couple of minor changes as means to regain the basic Links crawling functionality. These changes are documented in appendix A.4.

4.3.2.1 Links activity

The most basic criterion is based on a simple abstraction of the *gradient approach*. If there are no recorded recent changes in user's social connections, it is likely to remain like that. But if a user's recent Links reports greatly differ one from another, it is reasonable to assume that more changes are yet to happen. The Links activity is a measurement of recent changes within user's followers and friends list. After a full list of user's followers or friends has been collected by the TwitterEcho client and sent to the TwitterEcho server, that list is compared to the previously collected list from the same user. All the newly added users in the list and all the revoked users from the list are counted. This count represents the difference between two lists and it is the primary measurement tool for the Links activity.

The tracking of the Links activity is using almost the same procedure as the tracking of the tweeting activity.

1. The Links activity value is cooled off based on the time passed since the last Links check of that user.
2. The difference between the last two lists of followers/friends is cooled off based on the average time passed since every of the registered differences occurred. It is assumed that the occurrences of those differences were uniformly distributed in time, so the average time is the half of the time passed since the last Links check of that user.
3. The Links activity value is increased by the value calculated in step 2.

4.3.2.2 Tweet activity

Tweet activity is an indicator of activity borrowed from the Lookup service scheduler. The assumption behind this is that there might be a correlation between users' tweeting activity and changes in their connections with other users. This may especially be correct if users possess publicly visible Twitter accounts. In that case, they are potential targets for any Web crawler and can even be indexed by some of the popular online search engines. All this potentially leads to a larger followers reception – something worth investigating using the Links service.

4.3.2.3 Account lifecycle

Upon creation of a Twitter account, users are not following anybody and nobody is following them. After a few weeks/months, users' interests, relations and connections start to emerge and their followers and friends count start to increase. After the users have been active on Twitter for a couple of years, every user that might be interested in following them is already following. Users have reached a saturation point and not many more changes can be expected within their groups of followers and friends. Account lifecycle criterion for the Links scheduling revolves around the idea that the users should be crawled some time in between the account's initial period and the late, saturated period.

4.3.2.4 Mention and retweet activity

Mentions and retweets are special forms of addressing other users inside the tweet. Among other things mentions and retweets are simple ways for the users to redirect their followers' attention to other users or just something they tweeted. "Follow Friday", a popular topic on Twitter briefly mentioned in Section 3.1, is focused exactly on this kind of activity. Twitter users participating in "Follow Friday" take over the roles of the matchmakers, connecting some of their followers with some people they are following themselves ("followings").

Users that are mentioned or retweeted more are exposed to a much wider audience than just their followers. That is why it is likely that they will consequently attract more users to actually start following them.

4.3.3 Parameters

4.3.3.1 Inertia

Inertia parameter is a time duration expressed in seconds. It affects how much time is required to alter a record of member's activity pattern.

- Short *inertia* parameter (e.g., one day or less) – If a highly active user suddenly stops tweeting, soon the scheduler will “forget” about the user's previous activity and focus on the users that are currently more active. It pays off if a user permanently changed the tweeting activity because very few crawls will be wasted on that user. If a user was simply on a trip one day or sick for a few days that user will become active again and some of the tweets posted by that user will be overlooked because scheduler “forgot” the active users from several days before.
- Long *inertia* parameter (e.g., one week or more) – Completely opposite situation from previous one. The highly active users are difficult to be forgotten and it's harder for low activity users to earn activity ranking. One important incentive for using longer *inertia* periods is already stated in Section 4.3.1.5 and it indicates that by choosing a longer inertia period, users' activity can be modelled more precisely and much higher degree of differentiating between users can be achieved.

4.3.3.2 Maximum time without Lookup or Links

Max_time_no_Lookup and *max_time_no_Links* have a simple interpretation. If a certain time has passed since the last Lookup/Links call on a specific user then this user is automatically included in the list of users that are being handed to the client to perform the crawling. Theoretically, these two parameters should be set to infinitely high values. Users who have not been crawled for a very long time, have not been crawled for a very good reason. Because time after time, they continuously show no sign of activity. But, in the case of an inactive user the crawling of that user becomes exponentially less and less frequent until it comes to the point of no practical use. For

such practical reasons, it may be good to set these parameters to some relatively high value (*e.g.*, a month for the Lookup service, and several months for the Links service).

4.3.3.3 Starting activity values

Starting activity values could also be considered parameters on their own although they should be in relation to *inertia* parameter. If new users are added to a group of existing users, they need to be assigned the starting activity values. Since the scheduler has no previous data about the new users, statistically, the most correct assumption it can make is that these users are averagely active. Their activity is therefore calculated as the expected value from all existing activity values. But the question still remains what activity value should be assigned to a first "generation" of users. In that case some kind of common sense prediction should be made. Even though immanent error in judgment will only temporarily affect a scheduler's efficiency, this period of decreased efficiency can be shortened if users' assigned starting activity values imply **no less than 1 tweet per week** and **no more than 1 tweet per day**.

4.3.3.4 Service call frequency and client's workload

Service call frequency dictates how often the Lookup and the Links service are scheduled to be executed on one client. This parameter is set on each client separately as it simply tells a client how often to send requests to the server. Workload is the number of users that the server hands to a client each time a client requests a job. This parameter is set on the server side. Special care needs to be taken that job execution duration does not exceed the designated time available due to the service call frequency.

Other than that, setting these parameters is essentially a question of granularity. Whether to call the Lookup service 10 times in one hour and spend 15 Twitter API calls each time, or to call the Lookup service 30 times in an hour and spend 5 Twitter API calls each time? When using a single client, it is always better to have finer granularity *i.e.* more calls per hour since this allows a scheduler to check the same user more times in one hour. If more clients are running in parallel, it becomes impossible to have every client calling the server once per minute. The fine granularity is achieved by sheer number of clients, while one specific client is actually communicating with server as rarely as possible.

Incorrectly setting *service call frequency* and *client's workload* can cause too much stress for the client and/or the server. It can also cause insufficient utilization of clients. These are potential pitfalls that can consistently undermine crawler's efficiency.

4.3.3.5 Online time

Online_time parameter is a time duration expressed in seconds used in the "crawling of online users" described in Section 4.3.1.4. This approach tries to compensate for the biggest handicap of the REST API crawling approach. Many of the tweets posted within one minute are very difficult to be collected by the crawler using the Lookup service. This handicap is especially stressed when

a crawler is using one or two clients. Setting the *online_time* parameter requires a bit of tweaking in order to find the appropriate value.

- *online_time* period too short (less than 2 min) – Statistically it is unlikely that crawler will crawl enough online users exactly within this period since their last tweet.
- *online_time* period too long (over 30 min) – In a list of users the Twitter API is being queried for, the majority of users are picked because they satisfy the "online criterion". Small percentage of these users actually justify the increased attention. Too long *online_time* period also undermines the approach itself because it disables the crawler to look for other "candidates" that could be online.

4.4 Summary

In this chapter we presented the scheduling problem for the two different crawling services used to gather the Twitter data. The initial approach to dealing with this problem revealed some previously overlooked issues.

For the Lookup service the goal is to maximize the tweet collection rate while retaining similar coverage of all the users in a targeted community. To satisfy these goals new approach tracks the users' activities. It is assumed that this kind of information can help minimizing the waste of resources on unsuccessful crawls.

For the Links service four different criteria were suggested for keeping the social network connections charted with minimum discrepancy to the actual Twitter users' interconnections.

Scheduling Algorithms

Chapter 5

Evaluation and results

Series of experiments were carried out to evaluate the presented crawl scheduler.

All the tests were executed by running different scheduling algorithms on separate virtual machines on the same computer. These results are, therefore, subjected not only to the inevitable non deterministic factors like connection delays but also any possible minor irregularities occurring in sharing of computing power and resources. These effects can only influence the recorded scheduler's efficiency to a minor degree so all the results are credible enough to make certain conclusions.

The final section of the chapter includes evaluations that would provide more detailed information about the scheduler's characteristics but were not performed in the scope of this thesis.

Throughout this chapter we describe the five tests that were conducted during different evaluation periods.

Tests that were conducted include:

- a comparison between newly designed scheduling algorithm, the one within the current release of TwitterEcho and the baseline scheduler;
- testing the effects of different *inertia* parameters;
- experimenting with starting activity values;
- testing the effects of the *online_time* parameter;
- measuring the scheduler's efficiency in the distributed environment.

5.1 Comparing scheduling algorithms

First evaluation test was a parallel work of three separate crawlers with different scheduling algorithm. The three schedulers that were used are:

Evaluation and results

1. New scheduling algorithm – the new scheduling algorithm described throughout the thesis;
2. TwitterEcho scheduling algorithm – scheduling algorithm used in the current release of TwitterEcho crawler;
3. Baseline scheduling algorithm – pseudo-algorithm that is always selecting the users that were crawled longest time ago, users are checked in cycles so everybody is crawled equally frequent.

Parameters for the new algorithm were set as follows:

- *Inertia* parameter – 2 days;
- maximum time without the Lookup – 30 days;
- starting activity values – expected average activity of 1 tweet per 4 days uniformly distributed through all hours in a day;
- service call frequency & client’s workload – The Lookup service is executed once every 2 minutes spending 10 Twitter API calls on each call. This means a 1 000 users are crawled every 2 min;
- *online_time* – 3 min;

The current version of TwitterEcho crawler after more than a year of crawling collected the base of over 100 000 Portuguese users. Some of those accounts no longer exist because Twitter no longer recognizes their user ID. Prior to conducting any experiments, these users were deleted from the crawling list leaving only **87 978** users to crawl.

Result of this evaluation is shown graphically in Figure 5.1. Data points provided are number of tweets collected in 4 hour intervals in total time of 80 hours. While baseline algorithm shows roughly the same efficiency day after day, both scheduling algorithms show improvement the second and the third day of evaluation. The new scheduler produced the best results collecting **208 559** tweets against **115 617** collected by the scheduler included in the latest release of TwitterEcho. Baseline scheduler collected **70 882** tweets in the same period.

Since the new scheduler collected the most tweets, its data was used to test the severity of users’ activity differences stated in Section 4.1. Figure 5.2 shows the number of users that tweet more than the frequency stated on horizontal axis.

It is easy to notice a distribution resembling a characteristic *long tail*. Figure 5.2 is saying that out of 87 978 users that were being crawled during 3 day period, **57 144** of them posted something during that period and only **6 209** users were recorded to tweet more than one tweet a day.

TwitterEcho’s scheduler demonstrated lower tweet collection rate but that does not by itself imply it is worse than the new scheduling algorithm. The confusion table 5.1 shows that the new algorithm collected **123 060** tweets that TwitterEcho’s scheduler overlooked, but TwitterEcho’s scheduler collected **30 118** tweets that new algorithm missed. The number of the tweets that were

Evaluation and results

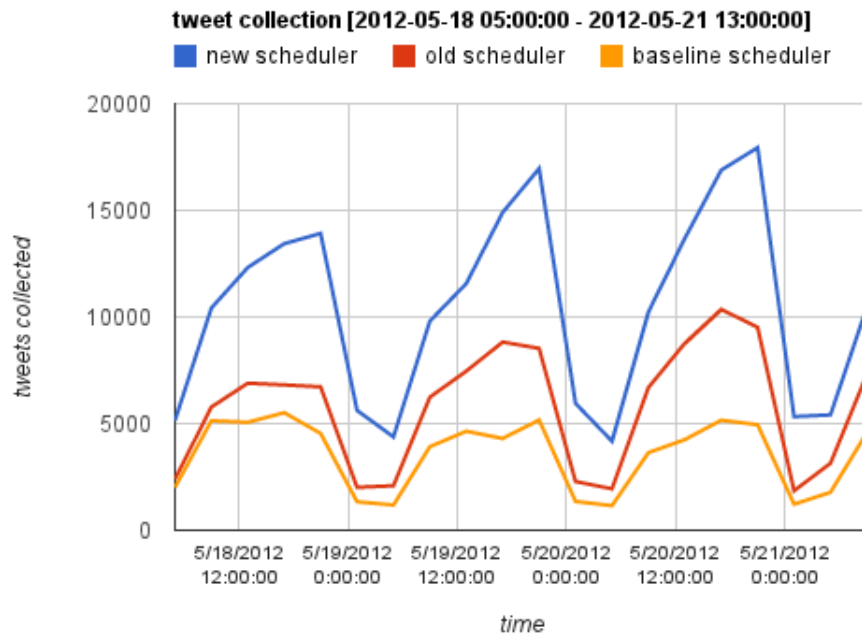


Figure 5.1: Scheduler comparison

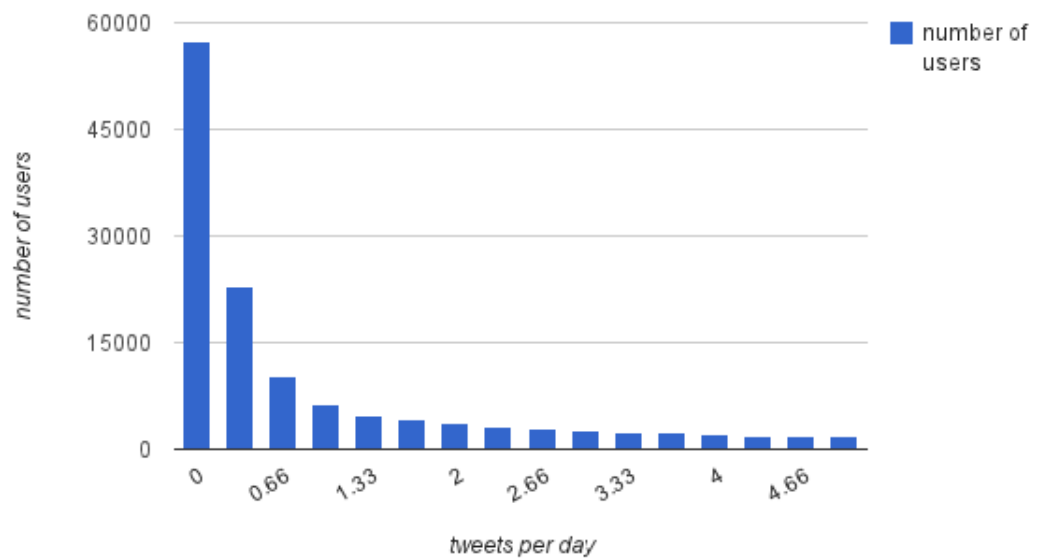


Figure 5.2: Activity representation of user base with 87 978 users

Table 5.1: Confusion table for tweet collection of both new scheduling algorithm and the one included in the latest TwitterEcho version

		Previous scheduler	
		Collected	Not collected
New scheduler	Collected	85 499	123 060
	Not collected	30 118	–

not collected by either of the crawlers is obviously unknown.

One of the most interesting things is that, during the evaluation period, all the tweets collected by the TwitterEcho scheduler were posted by **14 017** different users compared to **59 322** distinct users who were the authors of the tweets collected by the new scheduler.

Lists of top active users for both algorithms are shown in Table 5.2. Users’ names have been anonymized. The most active users from the perspective of the crawler are not necessary the most active users in reality. Large part of the lists in Table 5.2 are Portuguese radio stations informing the public about their program or news Web portals. What they have in common is the rate of tweet production that is more or less constant all the time. For this type of users it is relatively easy to get most of their tweets. On the other hand, for the crawling based on the REST API it is difficult to achieve high coverage of the users who combine long periods of inactivity with short periods of constant tweeting, replying and retweeting. This observation leads to an idea that these kind of users are also good candidates for inclusion in the streaming process.

Activity values formed over longer periods of time allow scheduler to make predictions about how many new tweets will be retrieved each call. Figure 5.3 shows the relation between scheduler’s predictions and actual results.

Figure 5.4 shows the amount of collected “conversational” tweets. *Online_time* parameter being set to 3 min, conversational tweets are all tweets posted within 3 min from the last tweet of the same user. Figure 5.4 is actually showing a conversational activity on Twitter over already stated period of three days. Although from the starting point scheduler managed to find more and more online users, a significant improvement is noticeable on 19th May 2012. On that day, a football Champions League final match between Bayern Munich and Chelsea took place in Munich at 19:45 UTC. On May 19th between 20:00 and 23:00, during the match, scheduler picked up 2 219 “conversational” tweets among which almost every fourth tweet contained some of the keywords connected to the football match (bayern, chelsea, drogba, champions, league, neuer, cech, muller, robben, ribery, lampard, final, luiz, schweinsteiger, matteo, campeões, torres, uefa, abramovic).

5.2 Testing *inertia* parameter

The second test was done by comparing three schedulers running the new algorithm with different *inertia* parameter:

- 1 day *inertia* period;

Evaluation and results

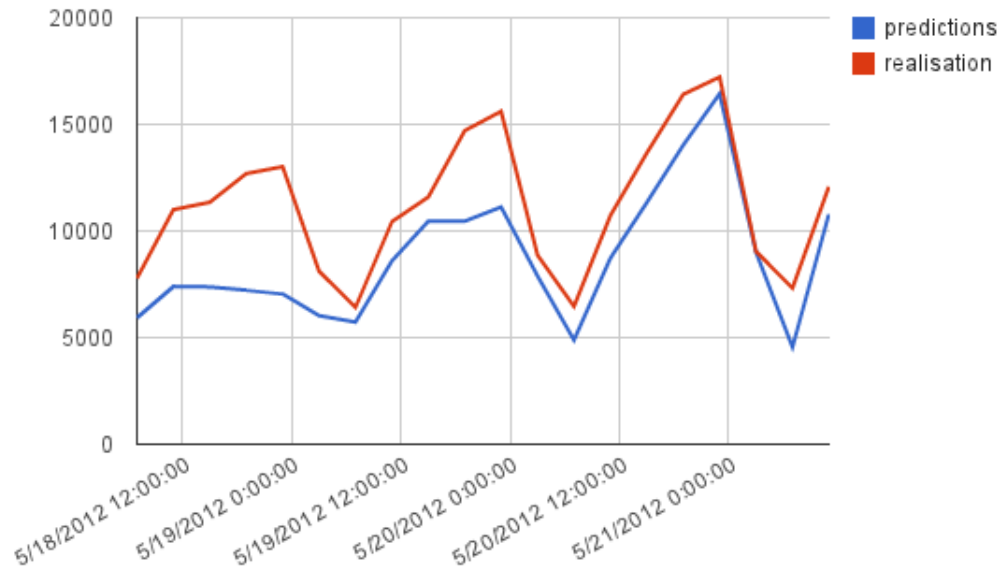


Figure 5.3: Scheduler's predictions vs. realisation

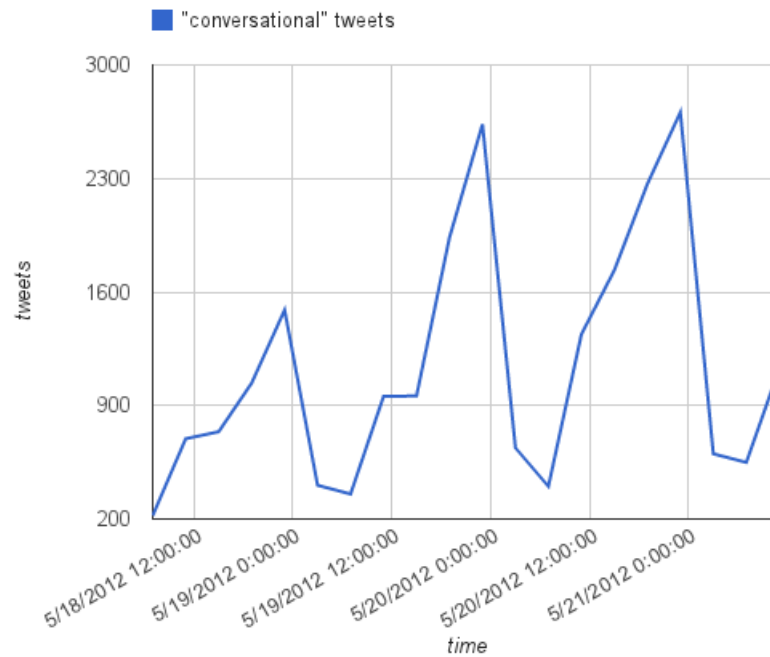


Figure 5.4: "Conversational" tweets

Table 5.2: Top active users registered by different algorithms

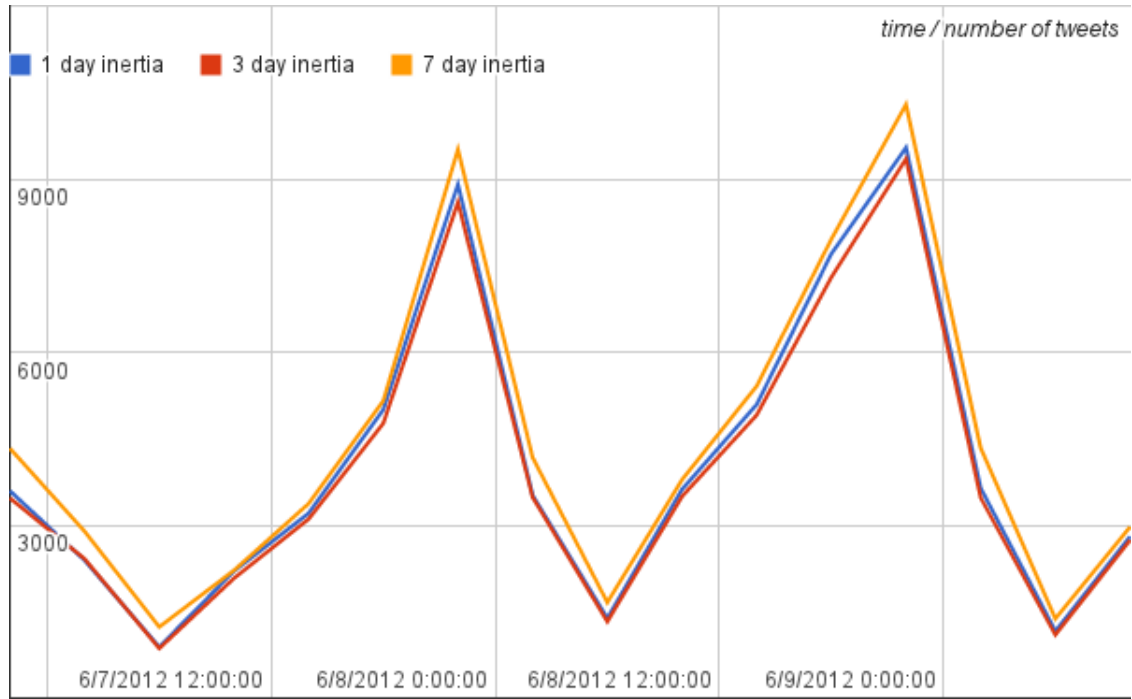
Previous scheduler		New scheduler	
Screen name	Tweets collected	Screen name	Tweets collected
user A	755	user A	1 062
user B	614	user C	1 002
user C	606	user B	889
user D	494	user D	652
user E	473	user E	603
user F	450	user K	593
user G	436	user F	563
user H	395	user G	509
user I	374	user I	427
user J	351	user M	422
user K	324	user H	414
user L	321	user N	411

- 3 day *inertia* period;
- 7 day *inertia* period.

Other parameters for the evaluation were the same for each of the evaluating versions:

- maximum time without the Lookup – 30 days;
- starting activity values – expected average activity of 1 tweet per 4 days uniformly distributed through all hours in a day;
- service call frequency and client’s workload – The Lookup service is executed once every 3 min spending 5 Twitter API calls on each call. This means a 500 users are crawled every 3 min;
- *online_time* – 0 min.

Although all three schedulers showed more or less similar performance during 40 hours of monitoring, their rankings stay constant during the whole time. The version with longest *inertia* period collected **70 650** tweets. The scheduler with 1 day *inertia* period collected 64 625, and 3 day inertia parameter seemed to be the worst with **62 440** tweets. It was expected that longer *inertia* will make finer distribution of activity points among users and by doing that the crawling could be more successful. The surprising fact is that the 3-day-inertia scheduler could not achieve that kind of superiority compared to the 1-day-inertia scheduler. In an attempt to find out why this happened, it was discovered that very low inertia schedulers have inherent mechanism of detecting online users even without using *online_time* parameter. When *inertia* period is short, cooling is faster and activity levels stay in low ranges. When a tweet is acquired, the increasing affects the surrounding activity values of the time in a day the tweet was created. That means t, thaif a relatively fresh new tweet is acquired, the activity is added both to a previous and to the next

Figure 5.5: Experimenting with *inertia* parameter

hour, so newly added activity will start accumulating immediately giving advantage to users who wrote most recent tweets. This may not be the case with longer *inertia* schedulers since activities range to higher values, making activity added based on recently collected tweets less significant in the total sum of activity points. The fact that gives some support to this theory is the observation that, during the evaluation, the 1-day-inertia scheduler collected **14 995** "conversational" tweets mentioned in Section 4.3.1.4, the 3-day-inertia managed to pick up only **14 017** and the 7-day-inertia **13 675**. Although the *online_time* parameter was disabled, for the sake of the argument a "conversational" tweet was considered any tweet posted within 6 min after a previous tweet by the same user.

Schedulers with different *inertia* period may have similar final results, but users' activity values look very different after some time of using different inertia values. Some users' activity values are shown on the figures 5.6 - 5.11. The users are anonymized for safety reasons.

The enclosed charts are picked specifically from users that showed similar pattern of tweeting activity over more days. *User #1* was mostly active in the afternoon (around 19:00), around midnight and in the morning with a smaller intensity. *User #3*, on the other hand, is a bit more consistent. Through all three days, this user was tweeting mostly around 01:00, and around mid day between 10:00 and 14:00.

Special attention goes to *user #2*, a user that seems to be tweeting 24 hours a day. This, of course, is almost certainly a bot – a computer program designed to post tweets automatically. This is not concluded simply because of user's tweeting frequency but also the unnaturally regular periods between tweets. Checking the content of the user's tweets one can see that these are all

Evaluation and results

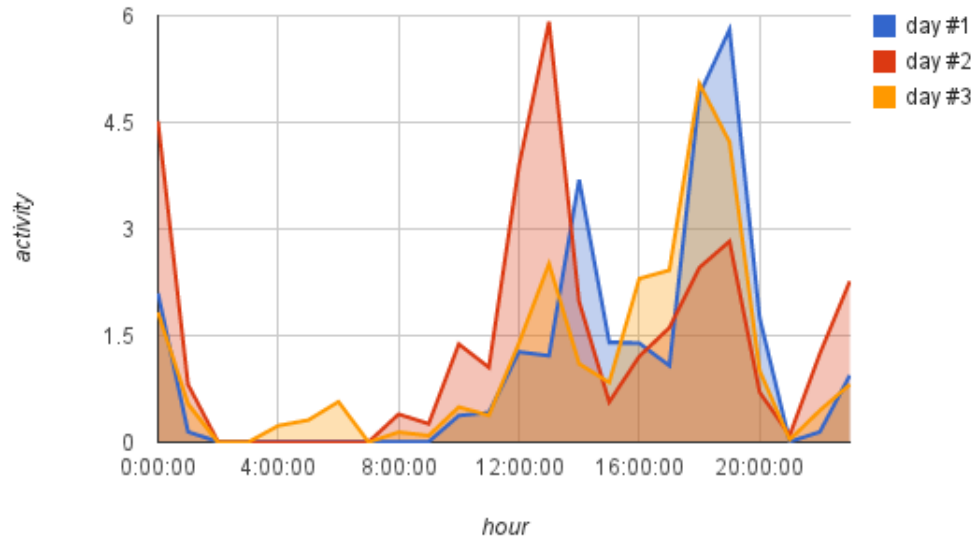


Figure 5.6: Activity values of the user #1 – 1 day *inertia* period

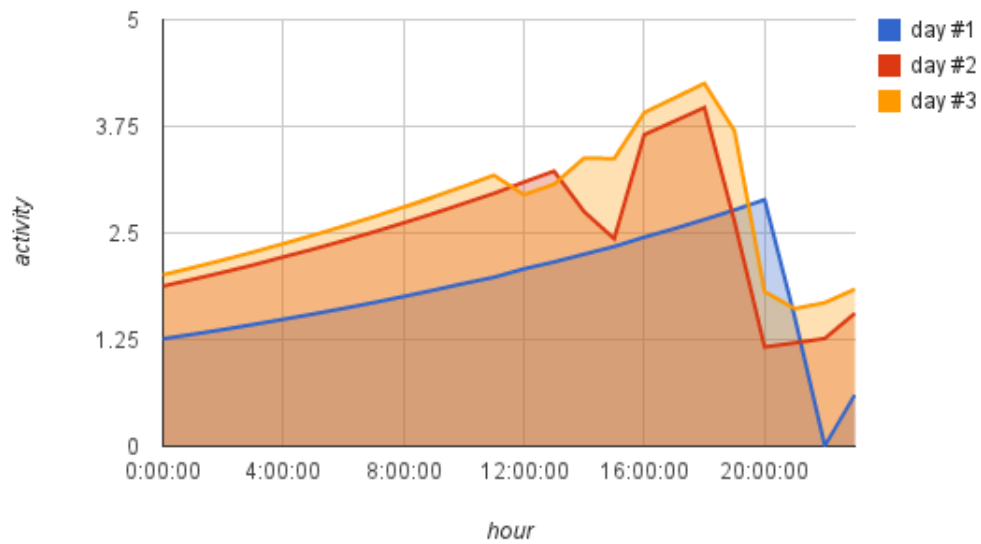


Figure 5.7: Activity values of the user #2 – 1 day *inertia* period

Evaluation and results

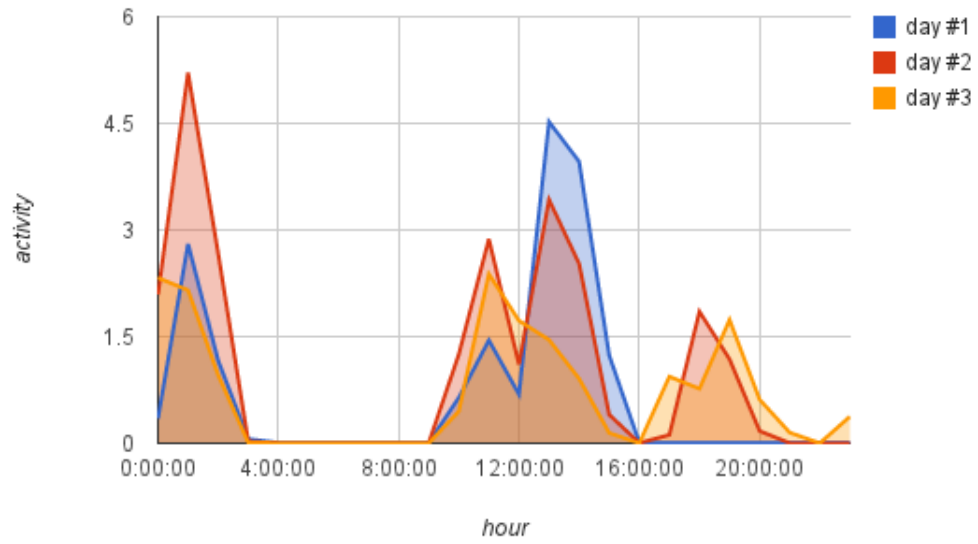


Figure 5.8: Activity values of the user #3 – 1 day *inertia* period

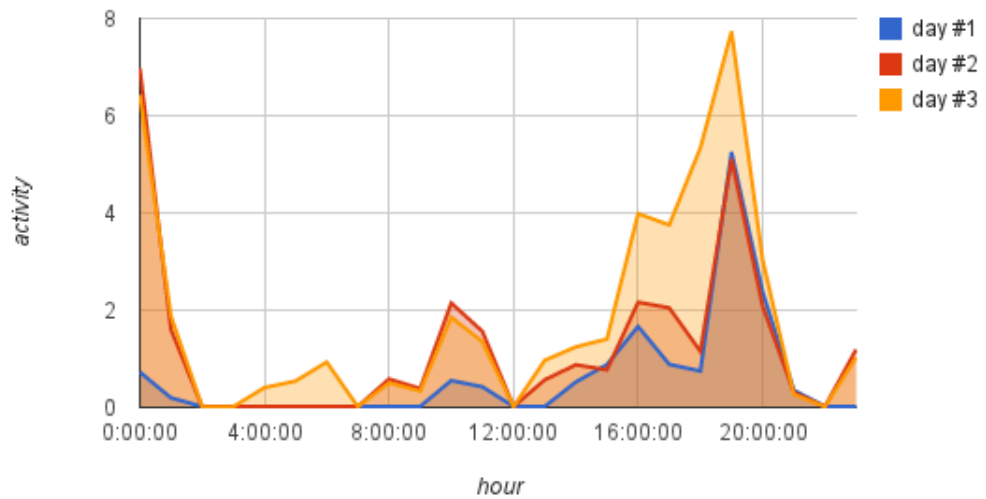


Figure 5.9: Activity values of the user #1 – 7 day *inertia* period

Evaluation and results

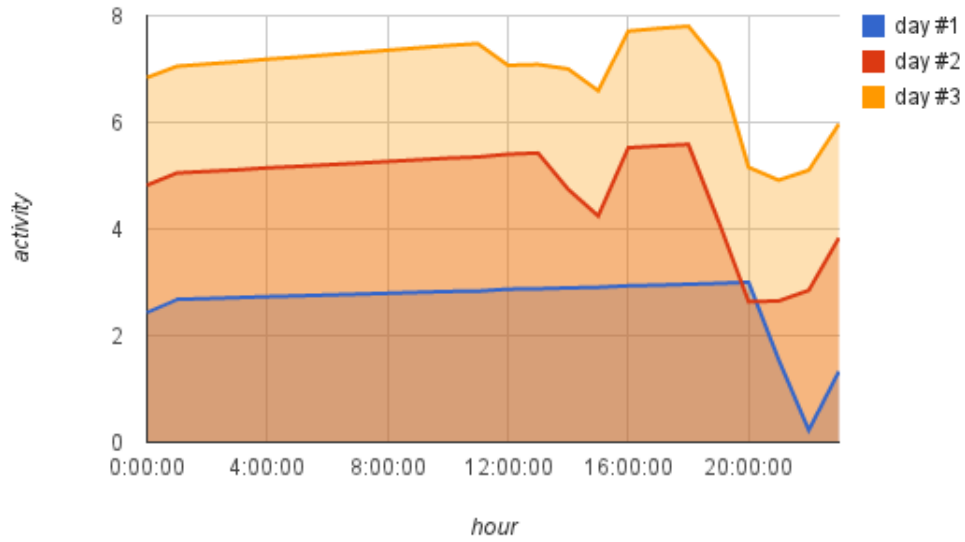


Figure 5.10: Activity values of the user #2 – 7 day *inertia* period

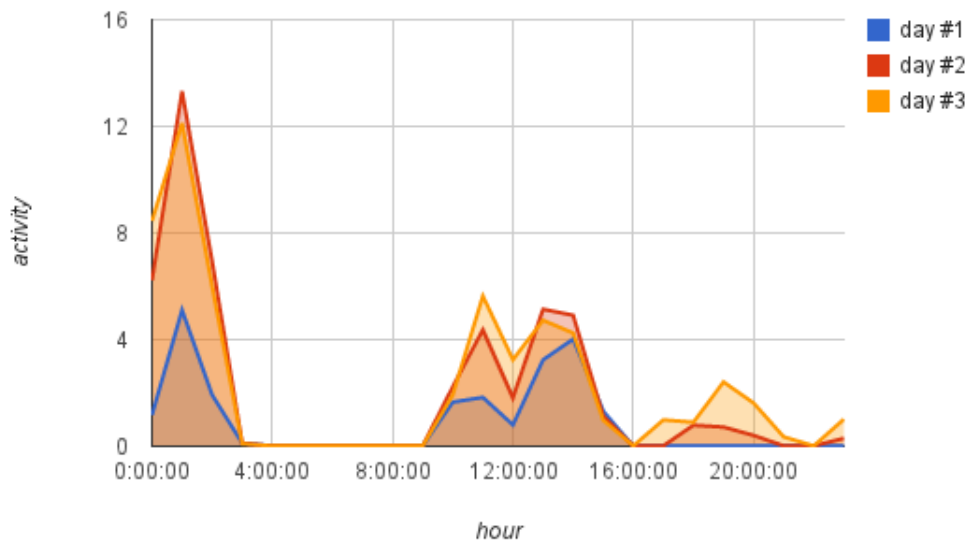


Figure 5.11: Activity values of the user #3 – 7 day *inertia* period

weather reports from a municipality called Figueira da Foz. Tweets are indicating temperature, humidity, atmospheric pressure, *etc.* on a regular basis.

Activity values reflect user activity during last *inertia* period of time. So, it is expected that, longer *inertia* parameter would have higher activity values and "smoother" activity curve.

5.3 Experimenting with starting activity

The third experiment aims to evaluate how the starting activity values affect the scheduler's performance. Setting starting activities to unreasonably high values causes a long period of similar treatment of both high active and low active users. This, of course, causes tweet loss. Three crawlers were started separately each of them conducting crawls according to the new scheduling algorithm only with their starting activity values uniformly distributed across all hours in a day but estimating different average tweeting activity.

- scheduler #1 – estimating 1 tweet per 2 days
- scheduler #2 – estimating 3 tweets per 2 days
- scheduler #3 – estimating 7 tweets per 2 days

Other parameters were set as follows:

- *inertia* – 7 days;
- maximum time without the Lookup – 30 days;
- service call frequency and client's workload – The Lookup service is executed once every 3 min spending 10 Twitter API calls on each call. This means a 1 000 users are crawled every 3 min;
- *online_time* – 0 min.

Figure 5.12 shows how setting the starting activities too high can have disastrous effects on crawler's performance. The scheduler that started with the activity values that imply expected average activity of 1 tweet per 2 days quickly dominated the other two schedulers which assumed higher activities for all users.

5.4 Effects of *online_time* parameter

Fourth test compared the effect of *online_time* parameter on the results of crawling. Three separate crawlers were put to work with differences only in *online_time* parameter. The first one had the tracking of online users disabled, the second one had the *online_time* set to 6 min and the third one was using 12 min period as *online_time* parameter. Crawlers were working simultaneously for three days (from 6th to 9th June 2012) with other parameters set as follows:

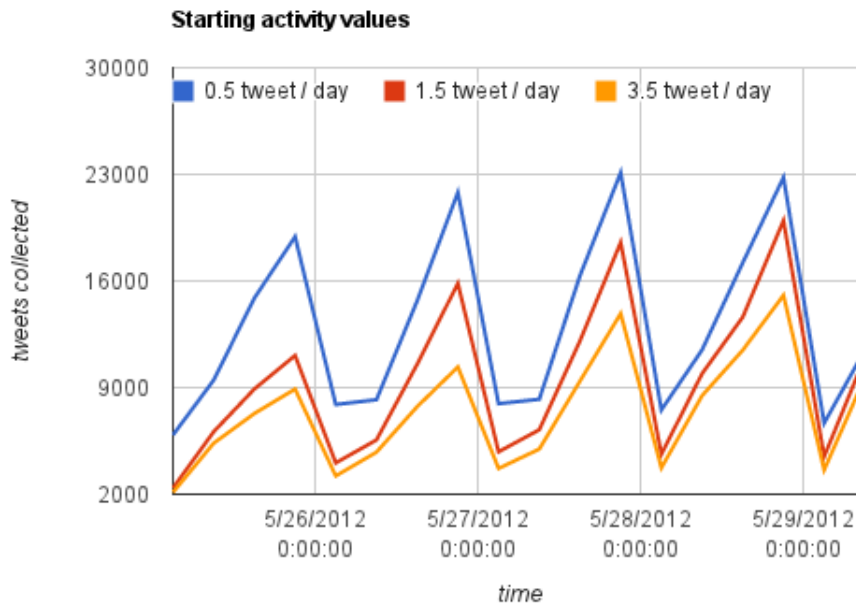


Figure 5.12: Experimenting with starting activity values

- *inertia* – 3 days;
- maximum time without the Lookup – 30 days;
- starting activity values – expected average activity of 1 tweet per 4 days uniformly distributed through all hours in a day;
- service call frequency and client's workload – The Lookup service is executed once every 3 min spending 5 Twitter API calls on each call. This means a 500 users are crawled every 3 min.

Results are shown in Figure 5.13.

Crawler with disabled tracking of online users collected **77 942** tweets, the one using 6 minute period as *online_time* parameter gathered just over **80 379**, and the one with 12 minute *online_time* period managed to acquire around **81 696** tweets. It is important to notice that at the beginning of evaluation, crawlers that were tracking online users showed much better results than the one that were not. After just one day, this difference started to fade away. One thing that remained the same is that crawlers using the *online_time* parameter always performed better than the competition during night time. In the times when general Twitter population are most active (evenings from 20:00 to 23:00) using too long *online_time* period can cause that the majority of users listed for crawling are chosen based on the "online criterion". This causes neglection of some other users known to be active at that specific time.

Evaluation and results

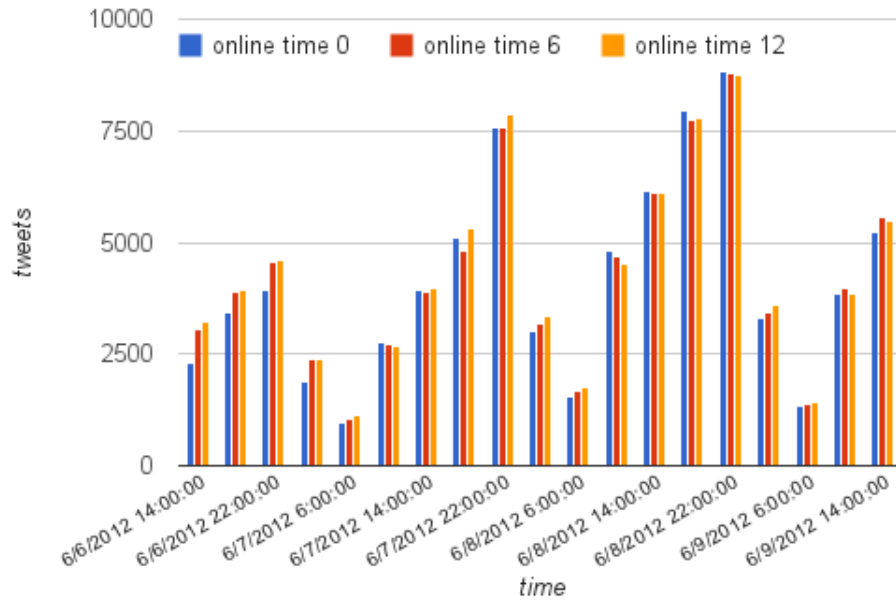


Figure 5.13: Crawlers using alternative values for *online_time* parameter

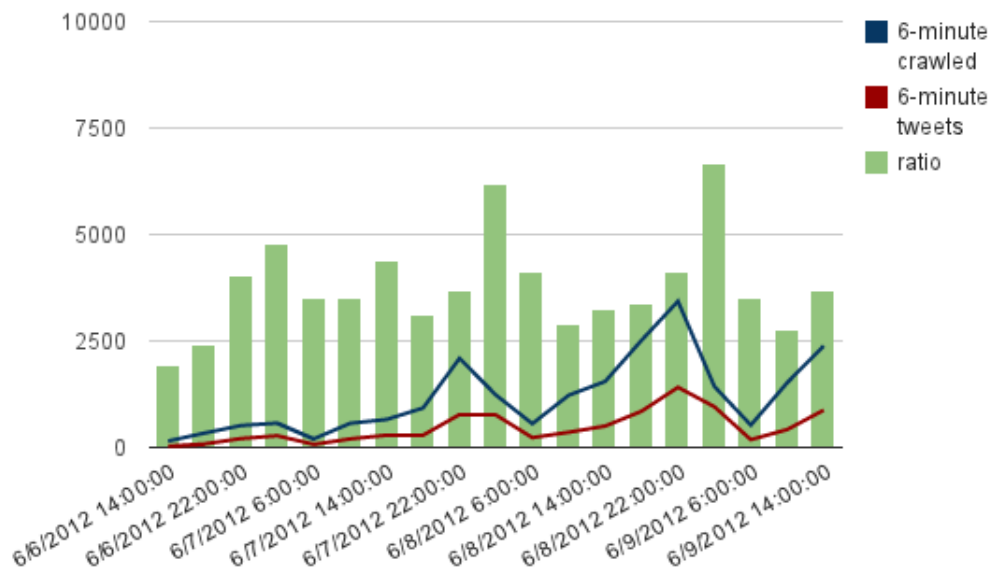


Figure 5.14: Ratio between the users selected for crawling based on the "online criterion" and the actual number of tweets retrieved from those users for 6-minute-online-time scheduler

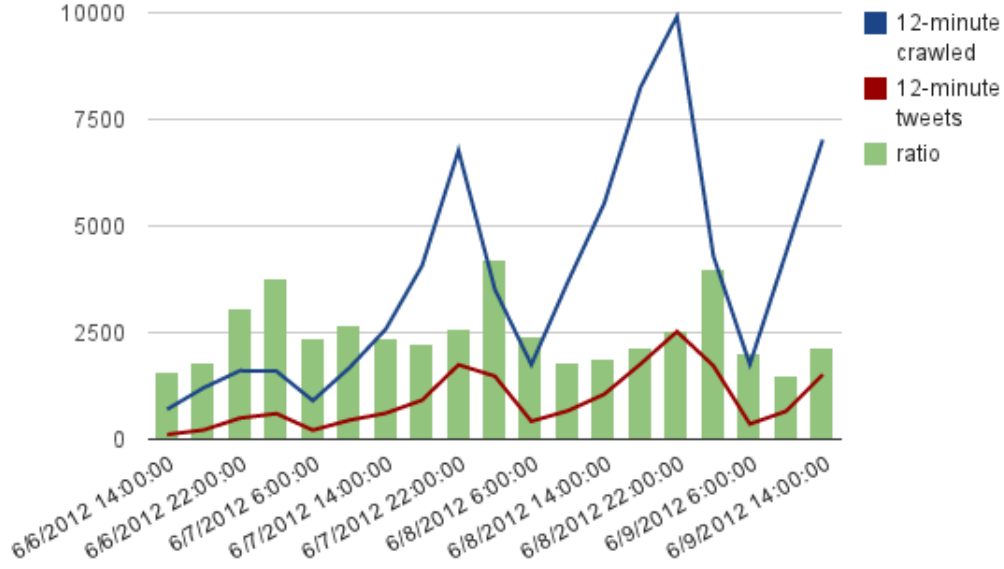


Figure 5.15: Ratio between the users selected for crawling based on the "online criterion" and the actual number of tweets retrieved from those users for 12-minute-online-time scheduler

In Figures 5.14 and 5.15 the columns reflect the percentage of the successful crawls in all the crawls performed based on the "online criterion". When using longer *online_time* period more tweets are collected through that criterion because more users are recognized as "online". However, the charts also show that the percentage of successful crawls is decreasing when the *online_time* parameter is increasing. A special care needs to be taken when setting the *online_time* parameter in order for this percentage not to go below the success percentage of ordinary crawling.

5.5 Efficiency in distributed environment

Fifth test shows the behavior of scheduler in the distributed environment using two and three clients. The purpose of this test is mainly to test the crawler's scalability. In theory, certain level of scalability should be assured by the ratio of the amount of the parallelizable work vs. the amount of non parallelizable work. In this case, parallelizable work is the crawling process itself: requesting and receiving data from the Twitter API. Non Parallelizable work is done by TwitterEcho server and it is mostly consisting of fetching and storing data from/into the database. According to the data collected in the first test (Section 5.1) an average client job consisted of around 40 s of parallelizable work and 2 s to 5 s of non parallelizable work. The reason for such variance in the length of the server work is the fact that MySQL relational database is used for data persistence. If the crawler becomes very efficient and starts retrieving a lot of tweets with each client call, the mere storing of tweets in the database can be accountable for over 90% of the

server's work. Nevertheless, this data shows that up to 7-8 Lookup clients should function well working in parallel.

Figure 5.16 shows the tweet collection rates during 6 days of crawling. The modifications of the crawler were done as follows:

- 9th June 2012 at 19:00 – crawling was started with only one client running;
- 11th June 2012 at 19:00 – one more client was started;
- 13th June 2012 at 21:00 – the third client was started;
- 15th June 2012 at 19:00 – the crawler was stopped.

The parameters that were used are stated here:

- *inertia* – 7 days;
- maximum time without the Lookup – 30 days;
- starting activity values – expected average activity of 1 tweet per 4 days uniformly distributed through all hours in a day;
- service call frequency and client's workload – The Lookup service is executed once every 3 min spending 5 Twitter API calls on each call. This means a 500 users are crawled every 3 min;
- *online_time* – 6 min.

The columns in the chart represent the difference from the number of tweets collected in the same time on the previous day. A drastic decrease in tweet number appeared on 14th June 2012. This anomaly can be explained by the severe increase of general Portuguese tweeting activity during the previous day between **15:00** and **19:00**. During this time Portugal's national football team played the European championship match against Denmark.

Results show an increase of the tweet collection rate with addition of extra clients. A tweet collection rate is usually not proportionate to the number of clients as it is already described in Figure 4.2. If tweet collection is nearly proportionate to the number of clients that means that the population that is being crawled is very poorly covered and that the tweet loss is high. In the opposite situation, if with addition of a new client, a tweet collection rate does not change noticeably that means that coverage is reasonably high and the tweet loss is not significant. The results also show that there is still place for improvement for crawling the Twitter population of this size.

However, the main purpose of this test was to evaluate the scalability of the crawler that is using the newly proposed scheduling algorithm.

During the already mentioned 4 hour long period of immense activity in the Portuguese "twit-tosphere" the crawler recorded only **358 s** of active work which consisted mostly of communication with the database. The server's job of admitting and dispatching HTTP packets is not included

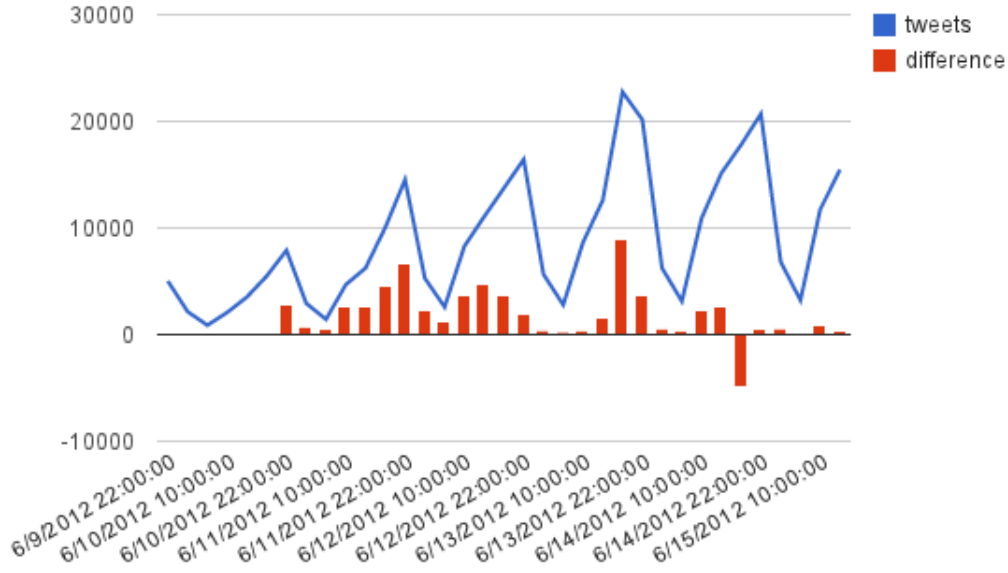


Figure 5.16: Tweet collection rates for variable number of clients

in this measurement. With the increased number of users the server’s workload would probably increase.

This insight shows that the crawler is very scalable and with increased number of clients should be able to provide sufficient coverage of a multiple times bigger Twitter population.

5.6 Other evaluations

In this section we suggest some ideas that would make the evaluation of the suggested scheduler more thorough.

The results of evaluating the scheduling of the Lookup service over longer periods would provide an insight into how the scheduler copes with the change of users’ activity patterns. The crawler running for a number of months usually registers that some people became inactive while some started tweeting a lot. After a long time of crawling it could be interesting to see how the users’ activity rankings have been changing over time and measure how much time did it take for the crawler to correctly position a recently noticed highly active or an inactive user in the activity rankings.

The methods stated in the thesis for the Links scheduling need to be extensively tested. The only sensible way to perform any test of the Links scheduler is to run multiple crawlers in parallel for a period of several months.

Using specialized, more "expensive" Twitter REST API services or streaming API services on a representative list of users, it is possible to estimate the exact coverage of the entire user base as the absolute measurement of crawler's efficiency.

5.7 Summary

Results have shown that the scheduling based on the suggested activity tracking methods preforms better than the current system used by TwitterEcho crawler. Regarding the parameters used by the scheduler, in short periods of evaluation starting activity values were distinguished as the most influential for crawler's efficiency. *Starting activity values* can only have a temporary deteriorating effect on the crawler's progress. After some time, the crawler establishes a level of efficiency designated by the actual users' activity on Twitter and other parameters like *service call frequency*, *client's workload*, *inertia* and *online_time* periods.

In the analysis of the crawler's scalability we established that the crawler with the included scheduling algorithm can run with a variable number of clients. Crawler's efficiency improves by adding more clients because this provides more Twitter API calls for the crawler to use. However, by calculating the ratio between the duration parallelizable work and the duration of non parallelizable work we determine that it is not recommendable to use more than 8 clients.

Evaluation and results

Chapter 6

Conclusion

In this final chapter we present what has been accomplished in the scope of this thesis. After that we discuss the properties and provide a quick review of the results. This chapter also includes some ideas about the future work – modifications that could potentially improve scheduler's efficiency.

6.1 Accomplishments

- A novel technique was implemented for tracking users' activities that is conceptually simple yet it provides much of the relevant information describing user's activity on Twitter.
- Activities measured with this technique were used for prioritization of users for the Lookup and the Links crawling.
- One additional approach was implemented for the Lookup crawling and several more for the Links crawling.

The scheduling methods proposed in this thesis track users' tweeting activities in order to arrange the crawling of the specific users at the times when they are most active. Results re-confirmed the importance of scheduling in the Twitter crawling. They also showed a significant improvement in tweet collection compared to the alternative scheduler which was used by the TwitterEcho crawler for more than one year. This improvement is a result of a detailed research of the scheduling problem that is explained in this thesis.

We believe that the presented approach to the scheduling has firm foundations because the required input is reduced to only a few parameters which are intuitive and the effects of changing their values are more or less predictable.

Newly designed scheduling algorithm shows excellent performance on crawling users that have consistent activity patterns and satisfying coverage of general Twitter population. The crawler's efficiency is constantly increasing for some time after it was started. This is the time required by the scheduler to "familiarize" itself with the users listed for crawling, to gather information about

Conclusion

them that will, later, be used for crawling prioritization. For this reason and due to the nature of the information being pursued, the evaluation of the scheduling algorithms requires several days for the Lookup scheduling and several months for the Links service.

The scheduler is not demanding in terms of computing power and it can be implemented on any Twitter crawler without noticeably affecting its scalability.

We hope that the scheduling algorithm developed as a part of this thesis will find its place in the future versions of the TwitterEcho crawler.

6.2 Future work

In this section we suggest some modifications for improving the scheduler's efficiency.

In general, during the weekends users display different activity patterns than on the weekdays. According to that, separate activity values can be stored for weekends and for weekdays. In that way the tracking of users' activities is more specialized and may lead to better user coverage.

In order to completely automate the crawler's work, the streaming service and the Lookup service need to have an established way of exchanging users. Streaming service is best utilized if it is used on a list of highly active users. The Lookup service should be used to crawl all the users that are not included in the streaming process. In that case, the Lookup service is covering much bigger population of Twitter users that can increase or decrease in size without interrupting the crawler's work. Keeping that in mind the Lookup service scheduler can every once in a while "nominate" the highest active users for streaming and the server can decide if some of the users included in the streaming process should be replaced with some more active users nominated by the Lookup scheduler.

In the end of the introductory part of Section 4.1 determining the ratio of the call frequencies for the Lookup and the Links service is presented as a special scheduling subproblem but it was not discussed in this thesis. Lookup scheduler's ability to predict the tweet collection rate at some periods in a day opens a space for implementing a dynamic exchange of resources between the Links and the Lookup service. If the Lookup service predicts that the scheduled call to the Twitter API will not return enough new data, this call can be skipped, thus saving valuable API calls which can be used subsequently by the Links service.

Appendix A

Implementation details

In this appendix we discuss the technical issues and describe the implementation of the new crawl scheduling approach. Diagram A.1 displays a structure of the TwitterEcho's database core. This diagram includes only the tables responsible for the crawling process and it does not include tables used for logging and error handling nor the ones used by any of the specialized modules within TwitterEcho research platform. The inclusion of the new scheduler into the existing TwitterEcho database model demanded some changes. These changes will be explained in the following sections. The modified database diagram is shown in diagram A.2.

In the following sections we discuss implementations of several aspects of the new scheduling algorithm, including:

- cooling the activity values,
- increasing activity values,
- accumulating activity,
- adjusting the Links crawling to support paginated responses from the Twitter API.

A.1 Cooling activity values

For the activity tracking purposes, we create the special *activity* table with 1-on-1 mapping to the *users* table. The new table, like all the changes to the original database structure, is visible in diagram A.2.

Following SQL update query is used for cooling off activity values right before a list of users is delivered to the client to perform a Lookup.

```
1 UPDATE `activity` SET
2 `activity0` = `activity0` * POW(e, (unix_timestamp(`activity`.`
    last_cooled`)-unix_timestamp(lookup_time))/inertia),
3 `activity1` = `activity1` * POW(e, (unix_timestamp(`activity`.`
    last_cooled`)-unix_timestamp(lookup_time))/inertia),
```

Implementation details

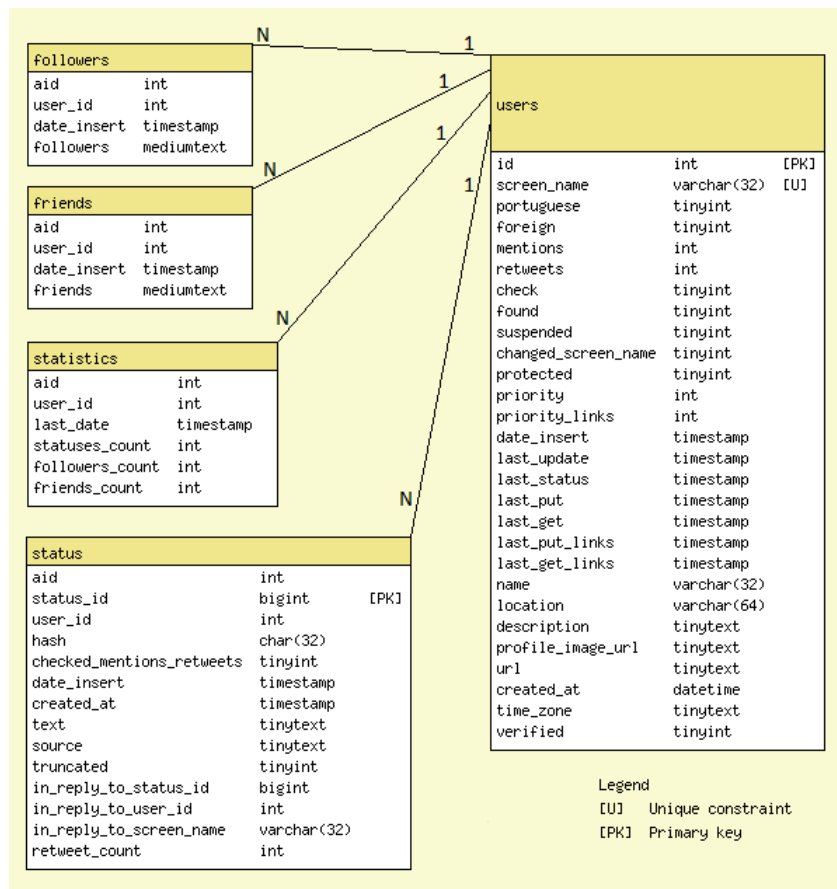


Figure A.1: The TwitterEcho's simplified database diagram

Implementation details

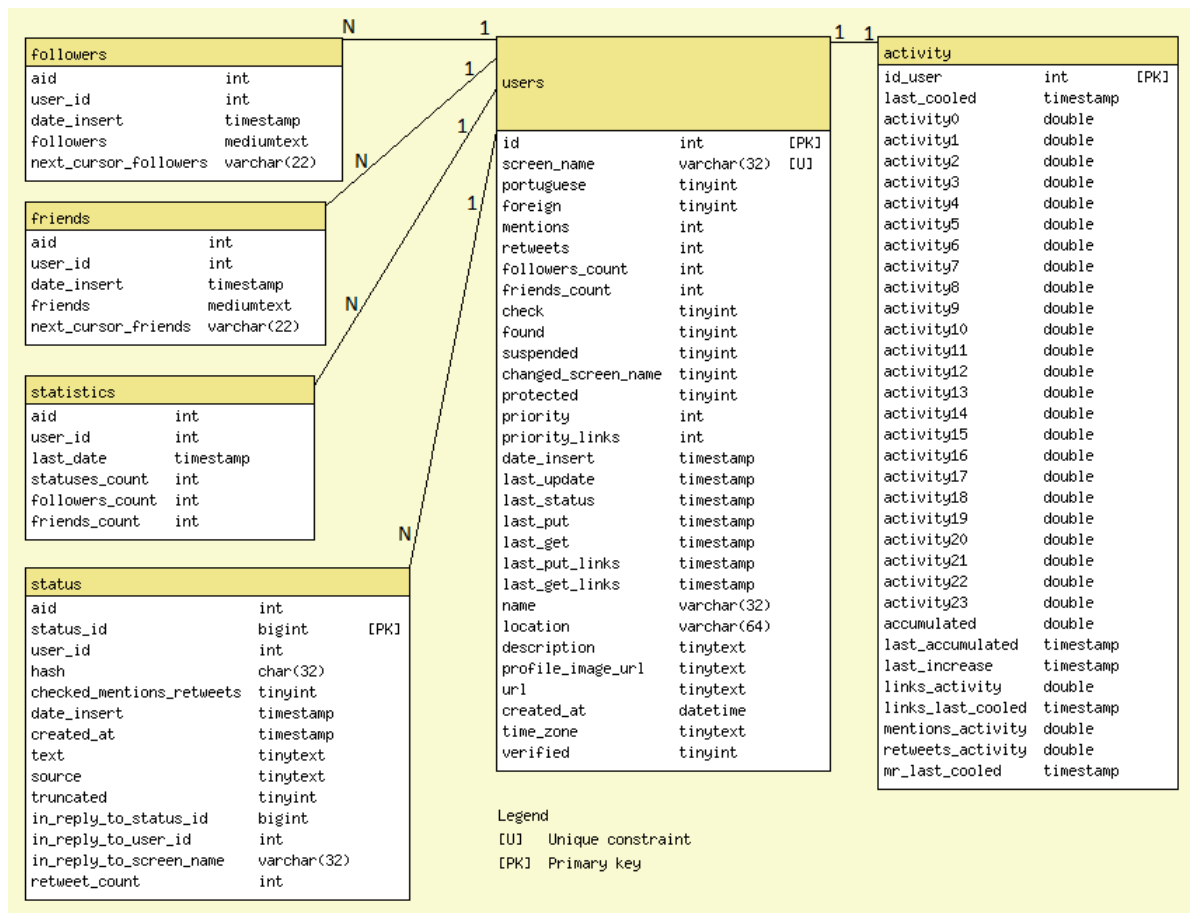


Figure A.2: The TwitterEcho's simplified database diagram after the implementation of the new scheduling approach

Implementation details

```
4 ...,
5 'activity23' = 'activity23' * POW(e, (unix_timestamp('activity`.`
    last_cooled')-unix_timestamp(lookup_time))/inertia),
6 'activity`.`last_cooled' = lookup_time,
7 'activity`.`accumulated' = 0
8 WHERE 'id_user' IN (list_of_users)';
```

Arguments needed for this SQL statement to be complete are:

- e – base of the natural logarithm
- $lookup_time$ – the timestamp of when the statement was executed
- $inertia$ – the $inertia$ parameter expressed in seconds
- $list_of_users$ – a list of users picked for crawling

A.2 Increasing activity values

Upon receiving a new tweet, the user's activity values increase around the time of day when the tweet was posted. Depending on that time, a certain amount of activity will be added to the previous hour mark and to the next hour mark. The actual ratio is determined using expressions A.1 and A.2.

$creation_time$ = the time when the tweet was created in a decimal number format
(e.g. 15 hours and 45 minutes is expressed as 15.75)

$previous_hour$ = $\lfloor creation_time \rfloor$

$next_hour$ = $\lceil creation_time \rceil \% 24$

$lookup_time$ = the time of the tweet's retrieval (decimal format)

$activity_x$ = activity value for hour x

$inertia$ = $inertia$ parameter expressed in hours and in a decimal format

e = base of a natural logarithm

$$activity_{previous_hour} \leftarrow activity_{previous_hour} + (1 - creation_time + previous_hour) \cdot e^{\frac{creation_time - lookup_time}{inertia}} \quad (A.1)$$

$$activity_{next_hour} \leftarrow activity_{next_hour} + (creation_time - previous_hour) \cdot e^{\frac{creation_time - lookup_time}{inertia}} \quad (A.2)$$

A.3 Accumulating activity

Accumulation of activity is performed for every user every time a client makes a request for a list of users to crawl. For this purpose we provide the entire function written in PHP code. Arguments are:

- *\$now* – the time when the function was called
- *\$last_accumulated* – last time when the function was called

The programming function is actually calculating for every user the area beneath the *line* on the chart representing the user's activity values and adding that value to the *accumulated activity*. The *lines* are formed by simply connecting the sequential dots (activity values) with straight lines. The following function shows a simple way to automate the generation and the execution of the SQL query needed to perform described task.

```

1 public function accumulateActivityForAll($now, $last_accumulated) {
2     $last_accumulated_ts = strtotime($last_accumulated);
3
4     $updateSQL = 'UPDATE `activity_view` SET `last_accumulated` = \''
5         . $now . '\', `accumulated` = `accumulated`';
6
7     $start = date('G', $last_accumulated_ts) + date('i',
8         $last_accumulated_ts)/60 + date('s', $last_accumulated_ts)
9         /(60*60);
10
11     $time_range = (strtotime($now) - $last_accumulated_ts)/(60*60);
12
13     $updateSQL .= ' + '.floor($time_range / 24).' * `activity_view`.`'
14         activity_sum`';
15
16     $time_range = fmod($time_range, 24);
17
18     $start_hour = floor($start);
19     $goal_hour = $start_hour + 1;
20     $goal = min($goal_hour, $start+$time_range);
21     $goal_hour %= 24;
22
23     while ($time_range >= 1/(60*60)) {
24         $v2 = $start - $start_hour;
25         $v1 = 1 - $v2;
26         $v4 = $goal - $start_hour;
27         $v3 = 1 - $v4;
28         $activityStart = 'activity' . $start_hour;
29         $activityGoal = 'activity' . $goal_hour;
30         $inc = ' + ' . ($goal-$start).' * 0.5 * ('.$v1.'*''
31             $activityStart.'` + '.$v2.'*''.$activityGoal.'` + '.$v3.'*`
32             '.$activityStart.'` + '.$v4.'*''.$activityGoal.'`)'';
33     }
34 }
```

Implementation details

```
27     $updateSQL .= $inc;
28
29     $time_range -= ($goal-$start);
30     $start = $goal_hour;
31     $start_hour = $goal_hour;
32     $goal_hour = $start_hour + 1;
33     $goal = min($goal_hour, $start+$time_range);
34     $goal_hour %= 24;
35 }
36
37 if (!mysql_query($updateSQL, $this->rec)) {
38     $this->saveError('activity->accumulateActivityForAll',
39         addslashes(mysql_error()), addslashes($updateSQL));
40     return false;
41 }
42
43 return true;
44 }
```

A.4 Links pagination

On 31st October, 2011 Twitter modified API services used for monitoring the relations between users. Among other things the two services that comprise the TwitterEcho's Links service ("followers/ids" and "friends/ids") were changed. The lists of followers and the lists of friends were divided into "pages" containing a maximum of 5 000 user ID.

A call to one of the services mentioned returns only one page. Each page has its own *cursor* which is like a pointer to that page. The cursor for the first page (the first 5 000 followers/friends) is always -1.

A response from the Twitter API contains the information stored in the JSON format. Along with the list of up to 5 000 followers/friends, response includes a cursor to the next page of followers/friends. A request sent to the Twitter API is also structured in the JSON format. Within the request it is possible to specify a user whose followers/friends are being queried and the cursor of the page being queried. If the final page has been reached the *next cursor* value included in the response from Twitter API is 0. It is unknown over how many pages the user's followers/friends span until the final page is reached. Therefore, it is possible to acquire a complete list of anyone's followers/friends but at the expense of an unknown number of Twitter API calls.

Two reasons that emerged out of "pagination" of the Links services make the Links crawling and scheduling slightly more complex:

1. It is impossible to know for sure if the remaining number of API calls will be sufficient to successfully crawl an x number of users. Successful crawl in the Links terms is a retrieval of the complete list of user's followers and friends.

Implementation details

2. It can also happen that some users have such a high number of followers that the 350 API calls per hour are not enough for one client to collect all of their followers and friends in one hour.

To minimize the frequency of situations when the client's remaining API calls prove insufficient to finalize the response we implemented the *Links cost estimation*. Links cost estimation checks the most recently collected number of user's followers and friends. According to the last record the Links scheduler determines the exact number of users that will be included in the next request to the Twitter API so that the restriction of 350 calls per hour is not breached. The number of user's followers and friends using the current release of TwitterEcho needs to be collected from the *statistics* table by sorting all the records by date of insertion and picking the most recent one. In order to improve the performance, we decided that the most recent followers and friends count should also be stored directly in the *users* table. This way a user's followers and friends counts can be accessed more quickly.

The most recent number of users' followers and friends is, in fact, just an estimation of the current number and some users are regardlessly impossible to be crawled by a single client in a reasonable time because their followers count is too high. The new database diagram (A.2) shows the new columns in *followers* and *friends* table – *next_cursor_followers* and *next_cursor_friends*. They provide a simple way for a client to leave a pointer to the next uncollected page of user's followers or friends. Another client can then crawl the same user without repeating the work already done by the previous client. The users with records in *followers* or *friends* tables with incomplete lists of followers/friends can be quickly recognized by having the *next_cursor_followers* or the *next_cursor_friends* values different than 0. These users are automatically chosen for crawling in every subsequent Links call until their lists of relations are finalized. The Links call of a specific user is completed only when both the user's list of followers and the user's list of friends are finalized.

Implementation details

References

- [Bar09] Ellen Barry. Protests in Moldova explode, with help of Twitter. <http://www.nytimes.com/2009/04/08/world/europe/08moldova.html?pagewanted=all>, April 2009. [Online; accessed 12-June-2012].
- [BE09] A. Burns and B. Eltham. Twitter free Iran: An evaluation of Twitter’s role in public diplomacy and information operations in Iran’s 2009 election crisis. *Record of the Communications Policy & Research Forum 2009*, pages 298–310, 2009.
- [BMRA10] F. Benevenuto, G. Magno, T. Rodrigues, and V. Almeida. Detecting spammers on twitter. *Proceedings of the 7th Annual Collaboration, Electronic messaging, Anti-Abuse and Spam Conference (CEAS)*, 2010.
- [BOM⁺12] Matko Bošnjak, Eduardo Oliveira, José Martins, Eduarda Mendes Rodrigues, and Luís Sarmento. Twitterecho: a distributed focused crawler to support open research with Twitter data. In *Proceedings of the 21st international conference companion on World Wide Web, WWW ’12 Companion*, pages 1233–1240, New York, NY, USA, 2012. ACM.
- [BRCA09] Fabrício Benevenuto, Tiago Rodrigues, Meeyoung Cha, and Virgílio Almeida. Characterizing user behavior in online social networks. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference, IMC ’09*, pages 49–62, New York, NY, USA, 2009. ACM.
- [Cul10] Aron Culotta. Towards detecting influenza epidemics by analyzing twitter messages. In *Proceedings of the First Workshop on Social Media Analytics, SOMA ’10*, pages 115–122, New York, NY, USA, 2010. ACM.
- [GTC⁺09] Lei Guo, Enhua Tan, Songqing Chen, Xiaodong Zhang, and Yihong (Eric) Zhao. Analyzing patterns of user content generation in online social networks. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining, KDD ’09*, pages 369–378, New York, NY, USA, 2009. ACM.
- [Hir12] Rachel Hirshfeld. Obama: ‘electronic curtain’ has fallen over Iran. <http://www.israelnationalnews.com/News/News.aspx/154005#.T9jxBtOweBs>, March 2012. [Online; accessed 13-June-2012].
- [HM09] M. Hurst and A. Maykov. Social streams blog crawler. In *Data Engineering, 2009. ICDE ’09. IEEE 25th International Conference on*, pages 1615–1618, 29 2009-april 2 2009.
- [Hua11] Carol Huang. Facebook and Twitter key to Arab Spring uprisings: report. <http://www.thenational.ae/news/uae-news/>

REFERENCES

- [facebook-and-twitter-key-to-arab-spring-uprisings-report](#), June 2011. [Online; accessed 12-June-2012].
- [KGA08] Balachander Krishnamurthy, Phillipa Gill, and Martin Arlitt. A few chirps about Twitter. In *Proceedings of the first workshop on Online social networks*, WOSN '08, pages 19–24, New York, NY, USA, 2008. ACM.
- [Kir10] Marshall Kirkpatrick. Twitter to sell 50% of all tweets for \$360k/year through Gnip. http://www.readwriteweb.com/archives/twitter_to_sell_50_of_all_tweets_for_360kyear_thro.php, November 2010. [Online; accessed 13-June-2012].
- [KLPM10] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. What is Twitter, a social network or a news media? In *Proceedings of the 19th international conference on World wide web*, WWW '10, pages 591–600, New York, NY, USA, 2010. ACM.
- [Mel11] Mike Melanson. Twitter kills the API whitelist: What it means for developers & innovation. http://www.readwriteweb.com/archives/twitter_kills_the_api_whitelist_what_it_means_for.php, February 2011. [Online; accessed 13-June-2012].
- [PO05] Sandeep Pandey and Christopher Olston. User-centric web crawling. In *Proceedings of the 14th international conference on World Wide Web*, WWW '05, pages 401–411, New York, NY, USA, 2005. ACM.
- [Sal11] William Saletan. Springtime for Twitter - is the internet driving the revolutions of the Arab Spring? http://www.slate.com/articles/technology/future_tense/2011/07/springtime_for_twitter.2.html, June 2011. [Online; accessed 12-June-2012].
- [SOM10] Takeshi Sakaki, Makoto Okazaki, and Yutaka Matsuo. Earthquake shakes twitter users: real-time event detection by social sensors. In *Proceedings of the 19th international conference on World wide web*, WWW '10, pages 851–860, New York, NY, USA, 2010. ACM.
- [Twi12] Twitter API official documentation. <https://dev.twitter.com/>, June 2012. [Online; accessed 12-June-2012].
- [WHMW11] Shaomei Wu, Jake M. Hofman, Winter A. Mason, and Duncan J. Watts. Who says what to whom on Twitter. In *Proceedings of the 20th international conference on World wide web*, WWW '11, pages 705–714, New York, NY, USA, 2011. ACM.
- [Wik12] Web crawler. http://en.wikipedia.org/wiki/Web_crawler, June 2012. [Online; accessed 14-June-2012].
- [WLJH10] Jianshu Weng, Ee-Peng Lim, Jing Jiang, and Qi He. Twitterrank: finding topic-sensitive influential twitterers. In *Proceedings of the third ACM international conference on Web search and data mining*, WSDM '10, pages 261–270, New York, NY, USA, 2010. ACM.

REFERENCES

- [WSY⁺02] J. L. Wolf, M. S. Squillante, P. S. Yu, J. Sethuraman, and L. Ozsen. Optimal crawling strategies for web search engines. In *Proceedings of the 11th international conference on World Wide Web*, WWW '02, pages 136–147, New York, NY, USA, 2002. ACM.

REFERENCES