Distributed Large-Scale Data Collection in Online Social Networks

Hariton Efstathiades, Demetris Antoniades, George Pallis and Marios D. Dikaiakos
Department of Computer Science
University of Cyprus
Nicosia, Cyprus
e-mail: [h.efstathiades, danton, gpallis, mdd]@cs.ucy.ac.cy

***Abstract*—The popularity and huge amount of information published in Online Social Networks (OSN) established them as one of the main data sources for a variety of research community fields. However, the design of a large-scale dataset collection campaign is a major problem for organizations and researchers who aim in addressing their research questions by analyzing this type of data. OSN platforms provide Application Programming Interfaces (API) to third party developers, which enable them to retrieve and use this data for applications deployment. However, due to OSN imposed limitations, the process of retrieving large scale data with the use of these APIs is challenging and time consuming, resulting in datasets which are either incomplete or outdated. It is relatively impossible for an individual scientist or research group to follow an efficient dataset collection procedure and build a large sample in a short amount of time. In this paper we present a framework for efficient crowd crawling of OSN. Our framework is based on the use of multiple OSN accounts, which are engaged in an efficient distributed collection process able to circumvent the imposed limitations without violating the terms of use. We present an evaluation of the proposed solution and demonstrate its performance in terms of dataset completeness and timeliness, for the case study of Twitter, one of the most popular platforms used in research.**

I. Introduction

The proliferation of Online Social Networks (OSNs) users over the last decade has enabled many scientific fields, such as social science and transportation [1, 13, 14], to drive into conclusions and validate their theories, through OSN data retrieval and analysis. For example, a variety of human behavior studies have been performed by social scientist due to existence of OSN data [24]. OSNs still experience a rapid increase in their daily user activity and population. Due to their user friendly approach, these platforms attract users who belong in different categories in terms of gender, age, education, job, geographic location etc. Users of such platforms are thus able to construct their social graphs, share content and interact with other users, as they would in their real lives, but in much larger scale and interaction range.

The large size of user directories and the frequency of interactions between them have established OSNs as one of the main actors in the era of "Big Data". The enormous number of messages that are published in a short amount of time, (Twitter users, for example, publish more than 500M tweets per day), along with the unstructured nature of these messages describe the 3Vs, of 'Volume', 'Velocity' and 'Variety' [25]. These characteristics introduce new challenges for anyone trying to acquire a complete and timely data set from OSNs. Considering also the data collection limitations imposed by OSN platforms themselves, the procedure becomes more challenging.

The design and implementation of a large scale data collection campaign over these powerful data resources is not a trivial task for an individual researcher or a single research group. The most popular platforms, such as Twitter, Facebook, LinkedIn, Renren, Foursquare, provide methods for retrieving data through specific Application Programming Interfaces (APIs). This functionality is useful for developers who aim in implementing an application on top of an OSN platform. Through such interfaces, researchers focus on collecting datasets in order to answer their research questions [5, 19, 20, 23]. However, due to the strict policies on API requests [18, 21], an efficient large scale dataset collection campaign, which results in a complete and timely sample, is a very challenging process.

Over the years many OSN platforms, like Twitter, have updated their API data request limitations. This action introduced new challenges in the field, as it has as direct result the deactivation of well-known and widely used data collection approaches [9, 15, 16]. Such methodologies were used to enable large scale crawling of the social graph by improving the collection throughput rate.

In this paper we design and introduce an OSN data collection framework that addresses the current challenges and provides the end-user with large scale crowd crawling capabilities. Furthermore, our proposed framework provides the ability of performing large scale dataset collection campaigns in the most efficient way, compared to state-of-the-art. We implement a crowd crawling prototype, using Twitter, and demonstrate its performance, with respect to OSN's request limiting policies.

The main contributions of this paper can be summarized as follows:

- We design and present a crowd crawling data collection framework, which enables an individual or a group of researchers to efficiently perform large scale data collection campaigns with the participation of OSN users. The proposed solution is able to efficiently: *a*) Collect historical data in an asynchronous manner, *b*) Retrieve the OSN stream in real-time.
- We implement a proof-of-concept prototype, which demonstrates the system under a case study on Twitter. We present an extended evaluation on different types

of devices along with a comparison over the state-of-the-art OSN data collection methods. We evaluate the proposed framework in both the large scale asynchronous data collection procedure and the collection of the real-time Twitter activity. Experimental results show that the proposed solution provides improvements in both data collection functionalities by more than 100x and 3x respectively.

The rest of the paper is organized as follows: Section II presents related work and challenges in the field of data collection from OSN; Section III presents the architectural design of the proposed framework; Section IV showcases and evaluates the system with an experimental study on Twitter and discusses the improvements over state of the art. Finally, Section V concludes the paper.

II. Related Work

The content that is generated through the interaction of users in Online Social Networking platforms is of high interest for the research community. However, the collection of a large-scale dataset is not a trivial task for researchers due to several challenges that are introduced mostly by the resources' limitations [3, 25].

Cho et al. [6] study the design of an effective web crawler. They present several problems in crawling procedure created by the rapid increment on the size of the Web. They propose multiple architectures for parallel distributed crawling framework and identify the challenges in the field of crawling the Web, similarly with [7]. Several challenges are also identified in the design and implementation of an effective large scale dataset collection framework, as the increasing quantity of information that is published in OSN platforms introduce relevant problems. The majority of these platforms maintain monitoring services to control the data throughput, introducing several additional challenges to the parallel data collection campaigns.

A major problem in a data collection campaign is the one of requests rate limiting policies of OSN providers. In the recent years major OSN platforms have used IP-based policies, which restrict a single machine to perform a certain number of requests [21]. The solution on addressing this challenge was straight-forward: a distributed data collection procedure was able to effectively overcome this limitation. Ding et al. [8] present the different categories of challenges for building a crowd crawling system, highlighting the resource diversity of the different parts, the different rate limiting policies from OSN providers, and the data fidelity. They propose a framework of crowd crawling, where a team of multiple research groups share resources in order to efficiently collaborate in a data crawling procedure. Their prototype is implemented over Planetlab, from which they take advantage of the availability of multiple nodes with different IP addresses.

Coalmine [27] is a social network data-mining system, which implements its own mechanism for collecting the data from Twitter and is able to retrieve data using the official API. Its overall architecture is based on distributed principles, where multiple IP addresses are used. Gjoka et al. [14] propose another similar framework for large scale dataset collection from Facebook. They design and implement a distributed tool which is able to overcome IP-based limitations and collect a large sample.

However, OSN platforms, such as Twitter, have changed the IP-based policy to Application-based. The latter restricts a single application from performing a large number of requests. Thus, this update makes a large scale dataset collection procedure more complicated, as the distributed design in the proposed fashion is not functional. In this paper we propose a framework which is able to overcome the newly introduced challenges in the field and perform a large scale data collection campaign in the most efficient way. Furthermore, a basic low-resource demanding configuration of the proposed solution enables the collection of more than 2M complete user information in one day, while state-of-the-art requires a much more resource demanding configuration to achieve similar performance.

III. Proposed Framework Design

In this section we present the design of the proposed framework, by introducing the basic components and their core functionalities. Furthermore, we describe the communication between the different components of the system, and how each one of them contributes to the goal of increasing the efficiency in a large scale dataset collection campaign.

OSN platforms, such as Twitter, Facebook and LinkedIn, represent information in similar abstractions. Each user has a unique identifier, usually of type *long*. Similar identifiers are assigned in messages posted by the user, like *Tweets* and *Posts* in Twitter and Facebook respectively. The connections between the users are retrievable as edge lists, which denote either a reciprocal (friend) or direct (follower) connection between two users. The retrieval of this information can be achieved either using OSN provided API or through Web Scraping. However, the terms-of-services of popular OSN prohibit the data collection through automatic Web Scraping [1] [2].

A data collection campaign can be either *i*) Resource Specific collection or *ii*) a Real-Time stream collection. The first case provides retrieval services for a specific resource (e.g a user's profile, a tweet or post), while the latter enables the sample collection of real-time information that is being published in the OSN. Our proposed framework provides two different services in order to enable both cases of a data collection campaign, which are 1. Resource Specific Data Collection, and 2. Real-Time Stream Collection.

A. Crowd Crawling: Building the Tokens Repository

As mentioned in section II, a newly introduced challenge in data collection procedures is the update of IP-based limitations to Application-based ones. For a complete presentation of our crowd crawling approach we first describe the traditional data collection procedure, established through the available OSN API. An individual who aims in using the services of an OSN API is required to create an application in the specific platform. In our case, the requests are related with data collection,

[1] https://twitter.com/tos, Twitter Terms of Service (Last Accessed: October 2016)

[2] https://www.facebook.com/apps/site_scraping_tos_terms.php, Facebook Terms of Service (Last Accessed: October 2016)

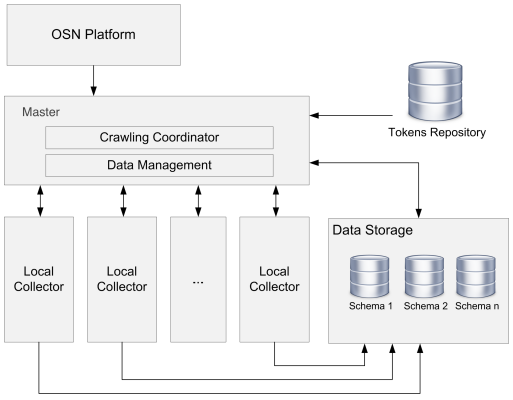

Fig. 1. General System architecture

either of a specific resource or the public OSN stream. By creating the application, the user agrees with the terms of service, and the OSN is able to monitor the requests executed through it. OSNs API policies restrict a single application of performing a large number of requests. In order to make authorized calls to an OSN API, each application must obtain a group of access *tokens* on behalf of a user, usually through the OAuth 2.0 specification [17]. These tokens are unique for each application and each user. For example, in the case of Twitter API, the platform provides four tokens; two are related with the application while the rest are related with the user who agrees to authorize the application to execute requests on user's behalf. Furthermore, the user has the ability to revoke the generated tokens at any time. This will result to the deactivation of the generated tokens; an action that will restrict from the application to execute API requests on user's behalf.

In IP-based limitations, the OSN API monitors the public IP address of the machine and applies the limitations per IP. Thus, a distributed data collection campaign in different machines, but with the same group of tokens, radically improves the procedure [4, 26]. However, after the latest updates, this is not possible as the majority of OSN API monitors the registered applications. As a result, even when an application (that utilizes the same set of tokens) is distributed in several machines with different public IP addresses, the limitations that apply are the same as if it was running on a single instance. Having this in mind, we proceed in a crowd-crawling approach, asking from OSN users to contribute to the data collection procedure by authorizing applications to access the API.

The procedure that we follow is the one suggested by the OSN platform. We first register an application in the OSN API. Then, we develop a service which asks from the users of our ego-networks (followers and followees) to authorize it to execute public data retrieval requests. Having the approval of the user, our service collects the generated tokens and stores them in a *Tokens Repository*. This repository contains a number of tokens that have been generated by OSN users. This crowd crawling procedure increases the number of tokens/resources that can be used during the retrieval process in the proposed system and takes place before the beginning of the data collection campaign.

Having multiple tokens enables us to activate a different set of them in order to avoid reaching the resources request limit; when we hit this limit the group of tokens becomes invalid for a certain amount of time t. We then move to the next group of tokens and execute the number of requests until we hit their limit. We follow this procedure repeatedly, until the condition $t - current\_time = 0$ is satisfied, as the group of tokens will become active again. Thus, with an n number of tokens we enable the continuous operation of the data collection campaign. This procedure is executed in each *Local Collector* instance, which is described in this section.

B. Resource Specific Data Collection

The proposed service provides functionalities related to asynchronous resource specific data collection. With this term we denote the procedure where we collect resource specific historical data enabling the retrieval and storage of all the available data of a user, given user's unique ID (UID). The proposed framework is able to crawl OSN platforms with the use of parallel API instances.

As depicted in Figure 1, the proposed system uses a Map-Reduce-like approach to overcome this limitation by partitioning tokens into a large number of small instances, greater than the available nodes, with some being replicated for performance objectives. Specifically, the system consists of three main components: (i) Master Component, (ii) Local Collector Component, (iii) Data Storage Component. The Local Collector Components are different instances running on different physical machines. Due to the latest API policies, which remove the IP-based requests limitations, multiple instances could be run on a single machine. However, the policies update to Application-based limitations increases the complexity in data collection process parallelization. The Master component is responsible to monitor and maintain the different Local Collectors, taking into consideration the resources demand and availability. The Master component assigns tasks to Local Collectors based on the provided UID list and the available tokens. Through the tokens and UID balancing, Master component manages to maintain the collector resources based on the demand. For example, if a Local Collector does not need the assigned resources, it returns them to the Master component which in turn assigns the resources to a more demanding Local Collector instance. Each Local Collector instance communicates with the Data Storage Component in order to store the retrieved data. The main task of the Data Storage component is to monitor the storage procedure and is able to perform modifications in the storage functionalities in order to ensure a maximum throughput rate. For every action it provides feedback to Master component and proceeds to modifications if needed (e.g. temporary store data in the file system, if the database engine is down).

UIDs retrieval: The Master component requires a list of UIDs to initiate the data collection procedure. Such a list can be retrieved from the proposed Real-Time Stream Collection service and/or through OSNs public directories. These directories are indexes to the public profiles of users, maintained by popular OSN platforms, such as Twitter[3] and Facebook[4]. The UIDs are used by the Crawling Coordinator, which initializes

[3]https://twitter.com/i/directory/
[4]https://www.facebook.com/directory

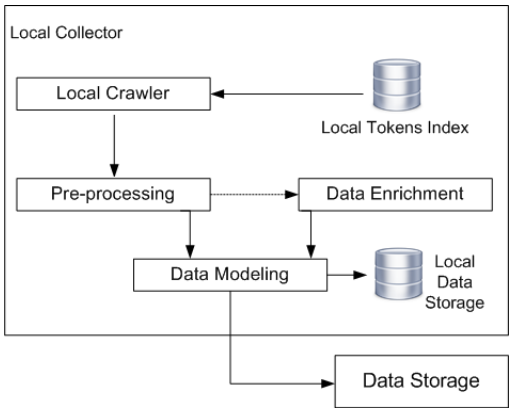

Fig. 2. Local Collector architecture

and distributes the crawling workload to the different Local Collectors.

Master Component: The Master Component is responsible for the workload distribution and monitoring of the data collection process. It has global knowledge about the system's state and maintains the resources based on the corresponding needs, by obtaining up-to-date crawling information from the different Local Collector instances. This component gets as input the list with UIDs that should be collected and calculates the load needed for each local instance. It then distributes the tasks and the resources based on the calculations. When a Local Collector requires more resources it sends a request to the Master component, which will then check the availability and update Local Collectors resources pool. On the other hand, when a Local Collector has reserved resources and does not need them, it notifies the Master component which in turn retrieves them back and makes them available for other resources. With this procedure, the system is maintained in a state where only the required resources are used, and each Local Collector has the highest available amount of the resources that it requires. In general, Master component has as goal to ensure that each Local Collector runs in full throttle 24/7, addressing API requests limitations.

Data Storage Component: The Data Storage Component is responsible for aggregating the anonymized results that have been collected from the different Local Collectors and store them with the most efficient way. This component is able to retrieve the data from multiple instances and store them in a central database. It maintains the storage queues and performs the necessary actions in order to ensure that it does not act as a bottle-neck. It is able to run real-time analytics and inform the Master component about the current metrics and actions taken. Such actions include the creation of different storage schema when the analytics suggest so, use of compression when running-out of space, store data in files when database engines fail etc. The component ensures that the retrieved data will be eventually stored in the file system in the most efficient way, at any given time. For the most efficient configuration, Data Storage and Master components should be deployed at the same machine.

Local Collector

In Figure 2 we present the architecture of the Local Collector component which is one of the main actors of the system. It is responsible for obtaining a task from Master component and perform the necessary actions in order to complete this task. Additionally, it provides real-time information about the progress to the Master component. Multiple Local Collector instances are distributed and deployed in different machines, increasing the efficiency and the throughput of the data collection system. Here we describe the internal components of the Local Collector and their tasks in the overall data collection campaign.

Local Crawler: In each Local Collector there is a Local Crawler, which is responsible to execute the requests to the OSN platform through the available API. This component uses the local tokens index which has been updated by the Crawling Coordinator of the Master component. Furthermore, it is responsible to maintain the crawling procedure in order to overcome the requests limitations, by requesting or dismissing API tokens. When the Local Crawler hits a request limit, it will automatically inform the Master component. If the Crawler Coordinator has available tokens, it will update the local index with the new tokens. On the other hand, if the Local Crawler is able to complete the assigned tasks with less tokens, it will inform the Master component and the Crawler Coordinator will dismiss the tokens and update the tokens index.

Pre-processing and Data Enrichment: The retrieved information is first passed through a pre-processing step, where is being cleaned and converted to the require encoding (e.g. convert the non-supported characters to unicode). Furthermore, during the pre-processing step, the retrieved data are being anonymized, according to privacy protection policies[5] and OSN APIs terms-of-service. During the anonymization procedure we replace the user and posts' ids with random numbers. A part of the data is then parsed by the Data Enrichment step, where the collected information is being enriched from external sources (e.g. a Tweet is being parsed to NLTK for sentiment analysis [2], or the post-code of a geo-tagged tweet or post is being identified).

Data Modeling: After, these two steps the resulted data is forwarded to the Data Modeling component, where the final formating applies. Each Local Collector divides the general task in multiple subtasks, that can be executed in parallel on the same instance. For example, when a Local Collector gets the task of collecting 100,000 Twitter users, it is able to execute the crawling procedure in parallel, by running 5 threads which each one collects 20,000 users. Following this procedure the Local Collector is able to take advantage of the local workload division in smaller subtasks and better monitor the crawling procedure. The Local Collector communicates with the Data Storage Component thought a socket. Through this socket, it sends the data that are handled by the Data Storage component and stored at the final step in a database schema. In order to reduce the communication cost, Local Collector is able to temporary store the retrieved data locally and proceed to bulk insertions.

Failure Resistance: Each Local Collector instance maintains a local data storage component, which is activated in cases of failures in the communication with the Data Storage component. Retrieved data are stored in this local component, until

[5]http://ec.europa.eu/justice/newsroom/data-protection/news/20150128_en.htm

the communication is restored and transfered to Data Storage.

C. Real-Time Stream Collection

A main limitation in OSN platforms API is the one of filtering their public stream. Twitter, for example, makes accessible only 1% of the total Twitter stream though the corresponding API [6]. Thus, when a researcher aims at collecting Tweets that are being published from a specific geographical area, e.g. the city of London, she will only be able to retrieve the set that does not exit the threshold of 1% of the total Twitter Stream. Furthermore, Morstatter et al. conclude that the results of using the Twitter Streaming API depend strongly on the coverage and the type of analysis of the study, and highlight the need of methods and frameworks that compensate the biases in these types of API [22].

The proposed framework supports Real-time Stream Collection, a service that is able to overcome these limitations and collect OSN platforms' stream in the most efficient way. This service provides the functionality to retrieve the public stream with 2 different options: (i). Given as input a geographic boundary box, (ii). Given as input a set of terms. For (i) it constantly listens to the stream of the area that lies in a specific boundary box, while for (ii) it queries the API for posts which contain the specific terms.

Master Component: Similarly to Resource Specific Data Collection service, the Master component is responsible for monitoring the overall procedure. It takes as input the target file and the Crawling Coordinator distributes the load in the different listeners. For example, in the case of monitoring the stream of a specific location, it takes as input the geographical coordinates of the under investigation area and divides it in a grid. It then distributes the different boundary boxes in a team of Local Collectors, giving them the subtask to collect the stream of a much smaller geographical area. Data Management component is responsible to receive the feedback from the Data Storage and proceed to the necessary actions.

Local Collector: The Local Collector receives a task from the Master component and is responsible to constantly listen OSN stream based on the rules received, using the OSN API. Such rules are a boundary box or a specific set of terms. A Local Collector is also responsible to monitor its resources and ask from the Master component to redistribute the API tokens, and thus the workload, if required. For example, a Local Collector receives a task to listen to the Twitter stream of a part of the city of London. During rush hours, this area gets crowded, thus the stream exits the limits of the Twitter stream API. At the same time another Local Collector is responsible to receive the stream of a part of Nicosia, Cyprus, which is much less crowded than API thresholds. Both Local Collectors report their monitoring results and request from Master component to redistribute the load. Master's Crawling Coordinator then assigns the resources of the less crowded collector to the crowded one, by creating a sub-grid, while it assigns a nearby Local Collector to the part of the city of Nicosia.

IV. Evaluation

For the evaluation of the proposed framework we developed a proof-of-concept prototype, following the design requirements presented in section III. We evaluate the proposed framework for both provided functionalities, *Resource Specific Data Collection* and *Real-Time Stream Collection* over several case studies on the Twitter platform. The choice of Twitter for the evaluation was motivated from the fact that the openness of this platform has attracted a large number of research groups to perform analytics and drive into conclusions using data retrieved from its data servers [25]. In this section we present the experimental setting and the results of the experiments. We then discuss our findings and compare with related work.

A. Properties of Interest

The proposed framework visits a Twitter user's account and collects the following information:

User profile: Each Twitter user is uniquely identified by his UID. In the public profile one can find information about the user's current status (description) and location. Additionally, in a user's profile additional automatic calculated fields can be found, such as tweets, followers and followees, profile creation date and profile image URL.

Tweet: a list of statuses are included in a user's Twitter account[7]. Each *Tweet* entry contains a unique identifier, the UID of the publisher, the text and a set of meta-data. Such meta-data include the timestamp, several flags that denote if the entry is geo-tagged, retweet, reply, favorite, if it has been retweeted and how many times, the number of mentions and hashtags contained and application that was used to get published. Moreover, for each geo-tagged Tweet, information about the geographical place is included, such as the country, country code, place name, street address, place type and a geographical boundary box. Furthermore, we enrich each geo-tagged Tweet with its corresponding post-code area.

Ego-network: a users ego-network contains a list of followers and followees unique UIDs. The followers list contains edges that are ending on user's profile, while followees list edges that start from the user's profile.

B. Experimental setting

Our proof-of-concept prototype has been developed in Java. For the data storage component we use MySQL, which is a widely used relational database management system (RDBMS). For the integration between the data collector and storage components we use the JDBC driver.

We deploy a Master component instance on a server with 4-core 2.5GHz processor and 24GB memory. The Master component initiates four different instances of Local Collectors on four different machines, running on 4-core 2GHz processor with 4GB memory each. Additionally, we showcase an experiment on a Raspberry Pi Model B low cost device [8]. This scenario evaluates the execution of parallel instances of Local Collectors, coordinated by one Master component running on the infrastructure of a research institution. The Data Storage component is deployed on the same machine with the Master

[6]https://dev.twitter.com/streaming/overview

[7]We are able to retrieve at most the 3,200 most recent Tweets for each user, due to Twitter API request policy.

[8]A single-core, low-cost device, running at 700Mhz with 512MB RAM. https://www.raspberrypi.org/products/model-b

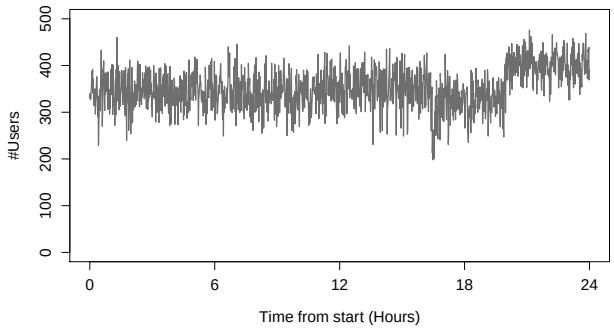

Fig. 3. Crawling throughput of an average *Local Collector* component for 24 hours. Each Local Collector is able to retrieve the complete set of *Properties of interest* for 397.2 users per minute on average.

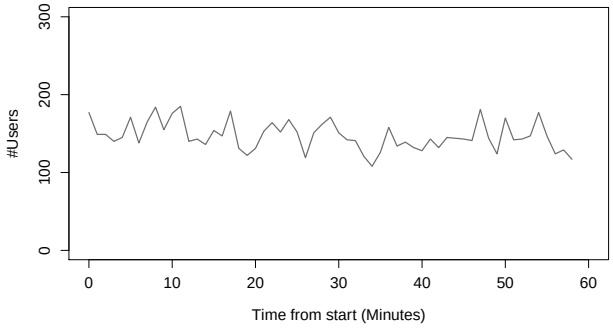

Fig. 4. Crawling throughput of one *Local Collector* component, running on a Raspberry PI low cost device. Each Local Collector is able to retrieve the complete set of *Properties of Interest* for 147.2 users per minute on average.

Users	Followers	Followees	Tweets	Places
2,300,574	1,220,972,850	635,276,364	1,612,766,674	1,040,240

TABLE I. NUMBER OF USERS, FOLLOWERS, FOLLOWINGS, TWEETS AND PLACES OF GEO-TAGGED TWEETS OF THE RESULTED DATASET.

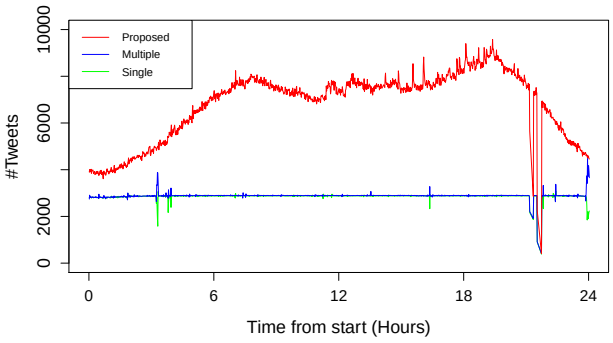

Fig. 5. Crawling throughput of stream listener, compared with single and multiple instances of Twitter API

component, in order to reduce the communication cost between these two actors.

Use Case Scenarios

Resource Specific Data Collection: In the presented crowd crawling case study scenario we use the online social networking platform of Twitter, one of the most widely used platforms in research. In order to generate the UID list we randomly sampled users from the dataset used in [19] and is publicly available. For each UID in this list, Local Collector instances request and store the complete *properties of interest*.

Real-Time Stream Collection: For this evaluation scenario we use the option of collecting the public stream of a boundary box. In order to get better insights on the performance we needed a scenario where the threshold of 1% of total Twitter stream will be exit. Thus, we give a boundary box with the complete world map, which indicates that we need to collect the public stream of all the locations. We then execute three different approaches in parallel: We collect the public Twitter Stream of this area using (i). *Single* Twitter stream listener using Twitter Stream API (ii). *Multiple* instances of Twitter API, listening to the same area, (iii). *Real-Time Stream Collection* functionality of the proposed framework. We then compare the results and present the insights.

C. Results

Crawling Throughput

Resource Specific Data Collection: We perform a distributed crawling procedure, following the described *Resource Specific Data Collection* methodology, for 24 hours. Figure 3 summarizes the throughput rate per minute for an individual Local Collector. Our proof-of-concept was able to obtain more than two million users during this period, having the four Local Collector instances collecting about 575,000 users each, without exceeding 9% of machines' memory usage. An average instance is able to collect and store more than 372 users per minute. During the collection procedure an instance collects the complete *properties of interest* of the requested users, as described above. The resulted dataset, presented on Table I, can be translated in more than 69GB of uncompressed data per Local Collector. Figure 4 showcases the performance of an instance running on a Raspberry Pi Model B. As we can see, in one hour of crawling, a Local Collector instance running on such device is able to collect the complete set of *properties of interest* of 8,820 users, a number which is by 3x higher than state-of-the-art [8]. These results show that the intelligent management of resources and tokens radically improves the traditional distributed methodologies.

Real-Time Stream Collection: Figure 5 summarizes the throughput rate per minute for the 3 different compared approaches. As we can see, the proposed system is able to perform a large-scale real-time monitoring campaign with up to 3 times higher throughput than the commonly used approach. The applied procedure on the proposed system resulted to the collection of more than 9M different Tweets, while at the same time Twitter Stream API does not return more than 3M. Furthermore, as we can see from the parallelized procedure, the *Single* and *Multiple Single* instances resulted to similar throughput, having the latter collecting 40K more unique Tweets.

D. Discussion

The evaluation of our proposed OSN dataset collection framework shows the feasibility of utilizing a number of OSN API Tokens, retrieved through crowdsourcing, to collect a complete and timely dataset, without violating the terms of use of the services. As shown above, through smart utilization of resource, an interested party can collect more data in minimal time, avoiding any bias in the research outcome, created by a long lasting data collection campaign. Additionally, through smart use of resources our framework triples the collection of the real time stream of OSN services. Such an increase can be valuable to both research and commercial application that react based on the real-time census of the active Online Social Network users. In addition to the evaluation performed in this section, the proposed framework was used for the data collection campaigns of [10–12].

V. Conclusions

This paper presents a framework for efficient data collection from Online Social Networks, enabled through crowd crawling of API data retrieval tokens. The proposed framework is based on the use of multiple OSN accounts, which are engaged in an efficient and smart distributed collection process, able to circumvent the imposed limitations without violating the terms of use. In all cases, the proposed solution proceeds to a pre-processing step where data are being anonymized, with respect to users' privacy and OSNs API terms-of-service. The evaluation of our proposed solution demonstrates its performance, in terms of dataset completeness and timeliness, for the case study of Twitter, one of the most popular platforms used in research. The presented framework enables the collection of more than 2.3M users in one day, retrieving also their Followers, Followees and Tweets. Furthermore, due to the intelligent use of resources, our framework triples the collection of the real-time stream of Twitter API.

VI. Acknowledgments

This work was partially supported by the iSocial EU Marie Curie ITN project (FP7-PEOPLE-2012-ITN).

References

[1] S. Asur and B. Huberman. Predicting the future with social media. In *Web Intelligence and Intelligent Agent Technology (WI-IAT), 2010 IEEE/WIC/ACM International Conference on*, volume 1, pages 492–499, Aug 2010.

[2] S. Bird. Nltk: The natural language toolkit. In *Proceedings of the COLING/ACL on Interactive Presentation Sessions*, COLING-ACL '06, pages 69–72, Stroudsburg, PA, USA, 2006. Association for Computational Linguistics.

[3] A. Black, C. Mascaro, M. Gallagher, and S. P. Goggins. Twitter zombie: Architecture for capturing, socially transforming and analyzing the twittersphere. In *Proceedings of the 17th ACM International Conference on Supporting Group Work*, GROUP '12, pages 229–238, New York, NY, USA, 2012. ACM.

[4] M. Bošnjak, E. Oliveira, J. Martins, E. Mendes Rodrigues, and L. Sarmento. Twitterecho: A distributed focused crawler to support open research with twitter data. In *Proceedings of the 21st International Conference on World Wide Web*, WWW '12 Companion, pages 1233–1240, New York, NY, USA, 2012. ACM.

[5] P. Burnap, O. Rana, M. Williams, W. Housley, A. Edwards, J. Morgan, L. Sloan, and J. Conejero. Cosmos: Towards an integrated and scalable service for analysing social media on demand. *International Journal of Parallel, Emergent and Distributed Systems*, 30(2):80–100, 2015.

[6] J. Cho and H. Garcia-Molina. Parallel crawlers. In *Proceedings of the 11th International Conference on World Wide Web*, WWW '02, pages 124–135, New York, NY, USA, 2002. ACM.

[7] M. D. Dikaiakos and D. Zeinalipour-Yazti. A distributed middleware infrastructure for personalized services. *Computer Communications*, 27(15):1464 – 1480, 2004.

[8] C. Ding, Y. Chen, and X. Fu. Crowd crawling: Towards collaborative data collection for large-scale online social networks. In *Proceedings of the First ACM Conference on Online Social Networks*, COSN '13, pages 183–188, New York, NY, USA, 2013. ACM.

[9] K. Driscoll and S. Walker. Big data, big questions| working within a black box: Transparency in the collection and production of big twitter data. *International Journal of Communication*, 8(0), 2014.

[10] C. Efstathiades, A. Belesiotis, D. Skoutas, and D. Pfoser. Similarity search on spatio-textual point sets. In *Proceedings of the 19th International Conference on Extending Database Technology, EDBT 2016, Bordeaux, France, March 15-16, 2016, Bordeaux, France, March 15-16, 2016.*, pages 329–340, 2016.

[11] H. Efstathiades, D. Antoniades, G. Pallis, and M. D. Dikaiakos. Identification of key locations based on online social network activity. In *Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2015*, ASONAM '15, pages 218–225, New York, NY, USA, 2015. ACM.

[12] H. Efstathiades, D. Antoniades, G. Pallis, and M. D. Dikaiakos. Users key locations in online social networks: identification and applications. *Social Network Analysis and Mining*, 6(1):1–17, 2016.

[13] L. Gabrielli, S. Rinzivillo, F. Ronzano, and D. Villatoro. From tweets to semantic trajectories: Mining anomalous urban mobility patterns. In J. Nin and D. Villatoro, editors, *Citizen in Sensor Networks*, Lecture Notes in Computer Science, pages 26–35. Springer International Publishing, 2014.

[14] M. Gjoka, M. Kurant, C. Butts, and A. Markopoulou. Practical recommendations on crawling online social networks. *Selected Areas in Communications, IEEE Journal on*, 29(9):1872–1892, October 2011.

[15] O. Goonetilleke, T. Sellis, X. Zhang, and S. Sathe. Twitter analytics: A big data management perspective. *SIGKDD Explor. Newsl.*, 16(1):11–20, Sept. 2014.

[16] A. Java, X. Song, T. Finin, and B. Tseng. Why we twitter: Understanding microblogging usage and communities. In *Proceedings of the 9th WebKDD and 1st SNA-KDD 2007 Workshop on Web Mining and Social Network Analysis*, WebKDD/SNA-KDD '07, pages 56–65, New York, NY, USA, 2007. ACM.

[17] M. Jones and D. Hardt. The oauth 2.0 authorization

framework: Bearer token usage. Technical report, 2012.

[18] S. Kumar, F. Morstatter, and H. Liu. Crawling twitter data. In *Twitter Data Analytics*, SpringerBriefs in Computer Science, pages 5–22. Springer New York, 2014.

[19] H. Kwak, C. Lee, H. Park, and S. 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.

[20] Y. Liu, C. Kliman-Silver, and A. Mislove. The tweets they are a-changin': Evolution of Twitter users and behavior. In *Proceedings of International AAAI Conference on Weblogs and Social Media (ICWSM'14)*, Ann Arbor, MI, USA, Jun 2014.

[21] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee. Measurement and analysis of online social networks. In *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, IMC '07, pages 29–42, New York, NY, USA, 2007. ACM.

[22] F. Morstatter, J. Pfeffer, H. Liu, and K. M. Carley. Is the sample good enough? comparing data from twitter's streaming api with twitter's firehose. *arXiv preprint arXiv:1306.5204*, 2013.

[23] S. A. Myers and J. Leskovec. The bursty dynamics of the twitter information network. In *Proceedings of the 23rd International Conference on World Wide Web*, WWW '14, pages 913–924, New York, NY, USA, 2014. ACM.

[24] D. Ruths, J. Pfeffer, et al. Social media for large studies of behavior. *Science*, 346(6213):1063–1064, 2014.

[25] K. Tao, C. Hauff, G. J. Houben, F. Abel, and G. Wachsmuth. Facilitating twitter data analytics: Platform, language and functionality. In *Big Data (Big Data), 2014 IEEE International Conference on*, pages 421–430, Oct 2014.

[26] A. H. Wang. Don't follow me: Spam detection in twitter. In *Security and Cryptography (SECRYPT), Proceedings of the 2010 International Conference on*, pages 1–10, July 2010.

[27] J. S. White, J. N. Matthews, and J. L. Stacy. Coalmine: an experience in building a system for social media analytics, 2012.