An Architecture for Collecting Longitudinal Social Data

Jeremy Blackburn and Adriana Iamnitchi
Department of Computer Science & Engineering
University of South Florida, Tampa, FL, USA
Email: jhblackb@mail.usf.edu, anda@cse.usf.edu

***Abstract*—As social computing reaches maturity, two research problems crystallize. On one hand, longitudinal studies of social media interactions become necessary for understanding, for example, how information and behavior diffuses through social networks. On the other hand, accurate representation of a person's social sphere requires aggregation of multiple sources of social data, such as online social networks, and interactions from email or instant messaging.**

This paper presents a platform for the collection, extraction and analysis of dynamic social data from multiple sources for use in observational social network research. Using an implementation of the architecture, we made daily observations of the cheating behavior of over 9 million gamers in a planetary scale online social network, finding new evidence that the spread of unethical behavior follows a contagion process and that a social penalty exists for publicly outed deviant actors.

I. Introduction

The explosion of online social interactions has led to a strong interest in interdisciplinary studies of human behavior. As social computing reaches maturity, two directions crystallize. First, longitudinal studies of social media interactions become necessary [1] for understanding, for example, how usage patterns relate to both the creation and uptake of user-generated recommendations, or how destructive unethical behavior spreads through social networks. Second, meaningful use of social knowledge requires the aggregation and processing of information from multiple sources of data, such as human interactions over diverse channels (OSNs, email, phone *and* blogs, for example) and sensor readings (i.e., location and speed). The promise of aggregated social information is manifested in enabling new classes of application that go well beyond the keeping-in-touch functionalities that made OSNs popular and into domains such as medical and social research, emergency interventions, collective awareness, and personalized service discovery [2].

These new directions, however, face a set of challenges. While collecting static snapshots of social networks has been somewhat understood [3], the collection of longitudinal observation faces significant scalability and adaptability challenges. When should the observations be made? How to deal with resource constraints for storing, transferring, and processing large amounts of data? Similarly, maintaining data aggregated from various sources raises its set of questions: what is a data format that allows the addition of new sources of information as they become available? What is a good data structure that allows for effective data processing in an ever evolving application ecosystem?

This paper presents a system for longitudinal acquisition of multi-sourced social data. In Section II we present our architecture. Section III discusses an implementation of the architecture used to collect over 500 million observations of the state of users in an OSN, and dynamically respond to changes in state in order to collect additional information. Related work is discussed in Section IV and we conclude in Section V.

II. Architecture

In earlier work, we collected information on over 10 million user accounts from Steam Community, the dominant OSN for PC gamers [4]. Over 700 thousand of the accounts had an algorithmically determined (by the company that runs Steam Community) cheating flag ("VAC ban") set. In the process of analyzing how cheating behavior spreads over time in the social network we encountered the following challenges:

- APIs provided by OSNs are often incomplete, poorly documented or simply not working, a natural outcome of a maturing platform and a growing user base.
- New functionalities are continuously added, and with them new attributes that prove relevant to data analysis. E.g., virtual goods inventories added between [4]'s crawls.
- The addition of new data and maturity of our understanding of new relevant questions to ask, leads to continuously changing needs both in data and analysis.
- New sources of data might present themselves. E.g., YouTube activity from accounts linked via a newly-introduced video upload feature.

We reached a preliminary architecture addressing these challenges using a three-prong approach:

Decoupled architecture: Decoupling of data collection from extraction and analysis enables our platform to be useful for studies of dynamic networks relying on volatile data sources. Most solutions employed in one-time social network analyses address data collection, extraction, and analysis as a monolithic component: the data analysis requirements inform data extraction, inevitably leading to multiple iterations of data collection or limited knowledge extraction. While decoupling data collection from extraction and analysis makes greater demands on storage, it allows for (a) successive iterations of data extractions as informed by data analysis results with minimum costs; (b) transparent collection of newly introduced attributes embedded within the data source independent of the data extraction or processing mechanisms; and (c) independent insertion or removal of new sources of data. This loosely

coupled architecture has the implicit benefits of fault tolerance and scalability.

Versioning and provenance: Longitudinal studies require revisiting the same data source over time to compare changes in the network. At minimum, this requires time stamping of data. More importantly, however, in light of the rapid growth experienced by social applications and the ever increasing ubiquitousness of Internet connected mobile devices, the format of data sources are likely to change over time, necessitating adaptations in data extraction code. In addition to data provenance, we are exploring the usage of existing distributed version control systems and continuous integration tools to provide knowledge of process provenance.

Multigraph, ego-net based model: Representing the fusion of multiple sources of social data for a single ego involves several design challenges. First, we observe that an undirected graph cannot adequately model contexts where relationships and interactions are not necessarily reciprocal in nature. Second, from a sociological perspective, even reciprocal relationships are usually not symmetric, calling for the ability to place a weight on an edge [5]. Third, simple graphs are unable to capture ego's ties *across* disparate social networks, and thus are insufficient for answering questions about behaviors that span application domains. For example, consider a study on diffusion, focusing on free form reviews given by gamers. Gameplay statics available via the Steam Community application, comments from YouTube, and participation in relevant communities such as `gaming.reddit.com` can be combined to form a more complete view of influence and exposure.

Finally, the abstraction of a multigraph stored as ego-nets allows for holes in the observational record to be patched in certain cases. Consider a situation where a change in state triggers the crawling of a user (see: Section III). In this scenario, information about the newly crawled user might be inferable from information collected on previous users, and vice versa, *without altering the provenance record*.

For example, say users *A* and *B* were crawled at times t_i and t_{i+n}, respectively. In *A*'s record no relationship between the two exists, yet in *B*'s one does. We can thus *infer* (at least partially) *A*'s neighborhood at t_{i+n} without actually collecting any additional information. We can also infer information about *B*'s neighborhood prior to having made any direct observations. Most importantly, the inference maintains the observational record; storing the graph as, e.g., an edge list, would eliminate the distinction between inference and direct observation, especially considering that the overwhelming majority of crawled datasets are not direct observations of the graph, but rather multiple observations of separate ego-nets.

A high level overview of our architecture can be seen in Figure 1. The scheduler (labeled 1) is responsible for issuing requests to crawl a particular user. The crawlers (2) receive the requests issued by the scheduler and retrieve raw data from various data sources. The output from the data collection subsystem is received by the storage process (3), which coalesces crawler output, for example the output from the Steam crawler and the YouTube crawler for a given user, and generates provenance metadata. Once coalesced, the storage process commits the raw data to the raw data document store (4). The extraction process (5) extracts meaningful social information, and stores it, along with additional provenance metadata, in the social graph store (6). Finally, the analytics process (7) operates on the structure exposed by the social graph.

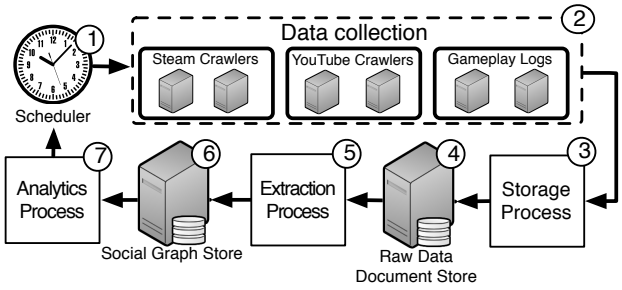

Fig. 1. A high level view of our architecture.

III. Ban History Implementation

The main purpose of the work in [4] was to explore the position of unethical actors (cheaters) in a planetary scale social network. With an initial analysis we discovered that cheaters' neighborhoods had a disproportionate amount of cheaters in them, and thus, there was a large connected component of cheaters. This initial analysis sparked a search for an explanation, and we next gathered approximate timestamps of when the cheating flag was set from a 3rd party source.

Although experiments using these timestamps provided evidence in favor of a contagion process, whereby cheating behavior spreads via friendship links in the social network, the data from the 3rd party was very low resolution, with an unknown collection methodology. Further, there were a fair number of cheaters we discovered in our initial crawl that had not yet been discovered by the 3rd party website, casting a shadow of uncertainty on our results, and precluding more sophisticated analysis.

A. Overview of Implementation

For the above reason, we set about building our own system for collection of ban history data using an implementation of the architecture described in Section II. Using newly available Web APIs [1] from the Steam Community we aimed to collect ban status history of an initial set of $\simeq$9 million users per day. Although not presented in this paper, the implementation will be extended to also include scans of YouTube accounts (discovered via a user's "videos" page) associated with monitored users.

Additionally, we used the analysis component of the architecture to schedule the retrieval of friends lists for newly banned users (2,337 in total with public friends lists during the observation period discussed in this paper) as well as a control set of non-cheaters (8,712 with public friends lists). The control set of users had their friends lists collected every day, while newly banned users had their friends lists retrieved on the day they were banned, as well as the next 10 days. Newly discovered users in the friends lists are added to the list of daily ban status scans.

[1] Since the time of our original crawls, Web APIs for querying the timestamped friends list of users as well as their ban status were added.

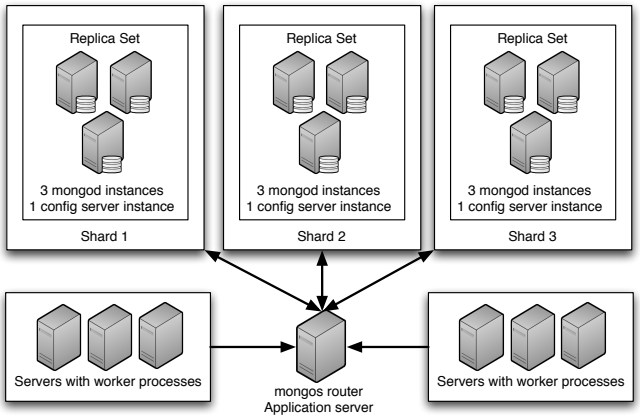

Fig. 2. An overview of the ban history implementation of our architecture.

Interprocess communication	Resque
Scheduler	`resque-scheduler`, `rufus-scheduler`
Data collection	custom Ruby code, `mechanize`
Storage process	custom Ruby code
Raw data document store	MongoDB cluster
Extraction process	custom Ruby code
Social graph store	MongoDB cluster
Analytics process	custom Ruby code, Ruby on Rails front end

TABLE I. SOFTWARE COMPONENTS USED IN IMPLEMENTATION.

The implementation, a bird's eye view of which appears in Figure 2, was straightforward and made heavy use of off the shelf software components. The major decision was the choice of data store backend. There has been some exploration of various database solutions suitable for social data storage [6], each with their own pros and cons. For our purposes, we settled on MongoDB. The choice of MongoDB was motivated by several factors:

- Easy to understand ad-hoc data modeling.
- Familiar SQL-esque querying and RDBMS-style indexing.
- Promises of high availability and performant scaling.

The application itself was built primarily in Ruby using many open source libraries, or "gems", as well as custom code. The Redis backed distributed job queue gem Resque was chosen for inter process communication. The `resque-scheduler` plugin was used for job scheduling, and uses `rufus-scheduler`, which provides `CRON` like functionality, as the underlying scheduling engine. The `mechanize` gem was chosen for accessing the Steam Community web services. The Ruby on Rails web framework provides a front end for the application, and the `D3.js` library provides some real time data visualizations. Table I lists all the software components we chose for corresponding pieces of the architecture.

B. Domain Specific Concerns

There were some issues related to the specific domain that are not addressed in the architecture described in Section II. First, the volume of data we were collecting hinted at the need for *delta based storage*. Even though the per-user data returned for each ban status query was small, in aggregate it added up fast, and so we implemented a simple delta storage scheme that found the last known value of each flag and stored the new value if it had changed [2]. A side effect of this was that any additional ban types added to the API were automatically integrated into our system [3].

The same basic approach was used for the storing of neighborhoods, but a more complex data structure was required:

```
neighborhood_observation = {
  user_id:string,
  observed_at:timestamp,
  plus:[friend_observations] <optional>,
  minus:[friend_observations] <optional>
}
friend_observation = {
  friend_id:string,
  relationship:string,
  friend_since:timestamp <optional>
}
```

The `user_id` field is a key representing the user this observation corresponds to, and the `observed_at` field is the time the observation was taken. The `plus` and `minus` fields contain the deltas computed from neighborhood observation, with the `plus` fields storing the edges that were added to this neighborhood, and the `minus` field those edges that were removed since the previous observation. `friend_observations` are edges, and labeled with the type of a relationship (at the time of this writing there is only one type of edge on Steam, "friendship"), and the `friend_since` field is the timestamp that the friendship was created.

Because MongoDB uses JSON as a record format, fields that were not used in an observation (e.g., if the user's neighborhood had no change) were not stored. The `friend_since` field in the friend_observation is optional for two reasons: 1) when someone is "unfriended", there was no need to store the date the relationship was created; we already had it on record in a previous observation, and 2) some friendships do not have a `friend_since` value available from the API [4].

The second issue is that the Web API is limited to 100,000 calls per day. Luckily, the ban status API allows for the querying of 100 users' ban statuses in a single call. Although our dataset grew slowly as friends were added from monitored users' friends lists, we were still left with several thousand unused API calls per day.

Unfortunately, the friends list API only returns a single user's friends list. To limit our calls, only the initial query of a user's neighborhood made use of the API and subsequent queries make use of the unmetered XML interface to Steam Community. A consequence of this is that the resolution of the `friends_since` field is reduced to 24 hours on all but the first observation of a user's neighborhood.

C. Results

We now present some of the results made possible with our implementation. We were primarily interested in finding more

[2] A timestamp recording that an observation was made is always stored.
[3] Two additional ban types, "community" and "trade", were discovered.
[4] Steam Community did not record the date of friendship creation until 2009.

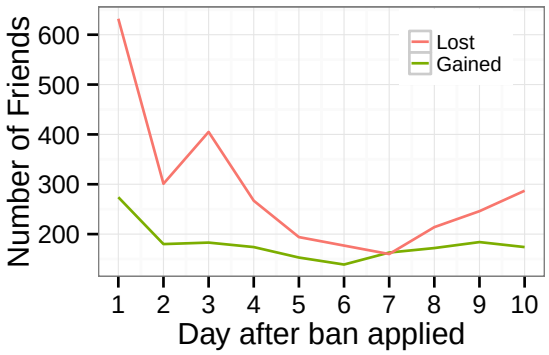

Fig. 3. The number of friends lost and gained for the 10 days following the cheating flag's application.

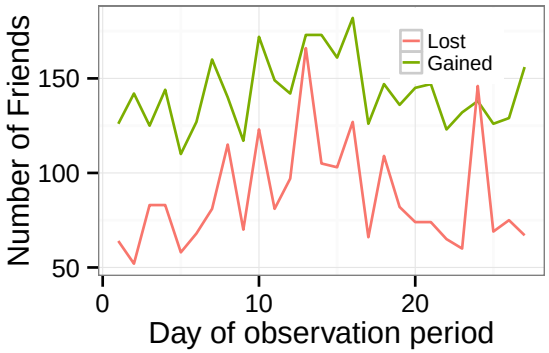

Fig. 4. The number of friends lost and gained by the control group over a four week period.

evidence to support a contagion theory of cheating behavior, however, we were also interested in the social impact of the publicly visible cheating flag.

We were able to confirm the observation that cheaters tend to lose friends after the ban is applied [4]. Figure 3 plots the number of friends lost and gained for the first 10 days after the cheating flag is applied, and Figure 4 plots the number of friends lost and gained by the control group over a four week period. There is a fair amount of activity in both sets of users' neighborhoods, but the cheaters' neighborhoods exhibit more flux. More importantly, the control group tends to gain more friends than they lose, while the cheaters lose far more friends than they gain. The new dataset made possible by the architecture presented in Section II has lead to the additional knowledge that this loss of friends occurs quickly, with the majority lost in the first few days after ban application.

Next, we make use of the high resolution dynamic data collected to visualize the contagion process. Figure 5 plots the progression of the 10 largest connected components in our monitored user set over the course of 30 days, divided into 5 day intervals. Only users that were in the monitored set prior to the end of an interval are plotted. Because users enter the monitored set only after a ban is applied, this visualization allows us to track the progression of cheating behavior through the network.

We see that although there are many disconnected nodes in the first interval, nodes in subsequent intervals come in attached to previously discovered cheaters, quickly forming clusters of cheaters in a classic diffusion pattern. This is very strong, large-scale, empirical evidence in favor of established theories of unethical behavior as a contagion [7], [8], [9].

IV. Related Work

Chau et. al appear to have published the first work on parallel crawlers for online social networks [10]. They describe a system that uses a breadth first search (BFS) and a centralized queue to assign users to a set of independent crawler tasks. Although quite similar from a high level view, Chau et. al's system was not designed to perform longitudinal data collection, and thus lacks a scheduler, an analytics engine, and thoughts on how to deal with a growing dataset over time (which we addressed by storing deltas).

Willinger et. al argued that the reliance on static datasets by OSN researchers was detrimental to our understanding of the real processes that govern human interaction [1]. By taking measurements at various points in time from a constructed "toy" graph with dynamic properties, they illustrate that overlooking this dynamism can lead to misleading or downright wrong results. Ultimately, they propose a multi-scale approach whereby graphs are analyzed at various granularities of spatial and temporal resolution.

Our earlier work [4] used a distributed BFS running on Amazon EC2 instances. The crawler instances were coordinated via an Amazon Simple Message Service queue and results were stored in a MySQL database. Although we performed multiple crawls, the system was not designed for longitudinal studies, lacked an integrated analysis component, and the choice of MySQL lead to complicated ad-hoc queries increasing the difficulty of exploratory data analysis. The experience we gained from our original crawler was the impetus for the system described in this paper.

A preliminary report by Souravlias et al. presents an architecture [11] with the same basic components as seen in Figure 1. The indication is that work towards an implementation is still being performed, but we suspect they will face many of the same challenges outlined in Section III.

V. Conclusion

This paper presented an architecture for dealing with some of the challenges facing researchers as we move beyond simple social network analysis. The three-pronged approach to design results in a decoupled architecture, facilities for versioning and data provenance, and a multigraph ego-net based data representation. When taken together, our architecture serves as a general framework for large scale observational studies of dynamic social networks with the ability to capture and aggregate data from multiple sources, and make them available for analysis in real time.

We demonstrated the feasibility of this architecture by building a prototype implementation. We used the prototype implementation to gather daily information on the cheating flag for over 9 million Steam Community users, responding to

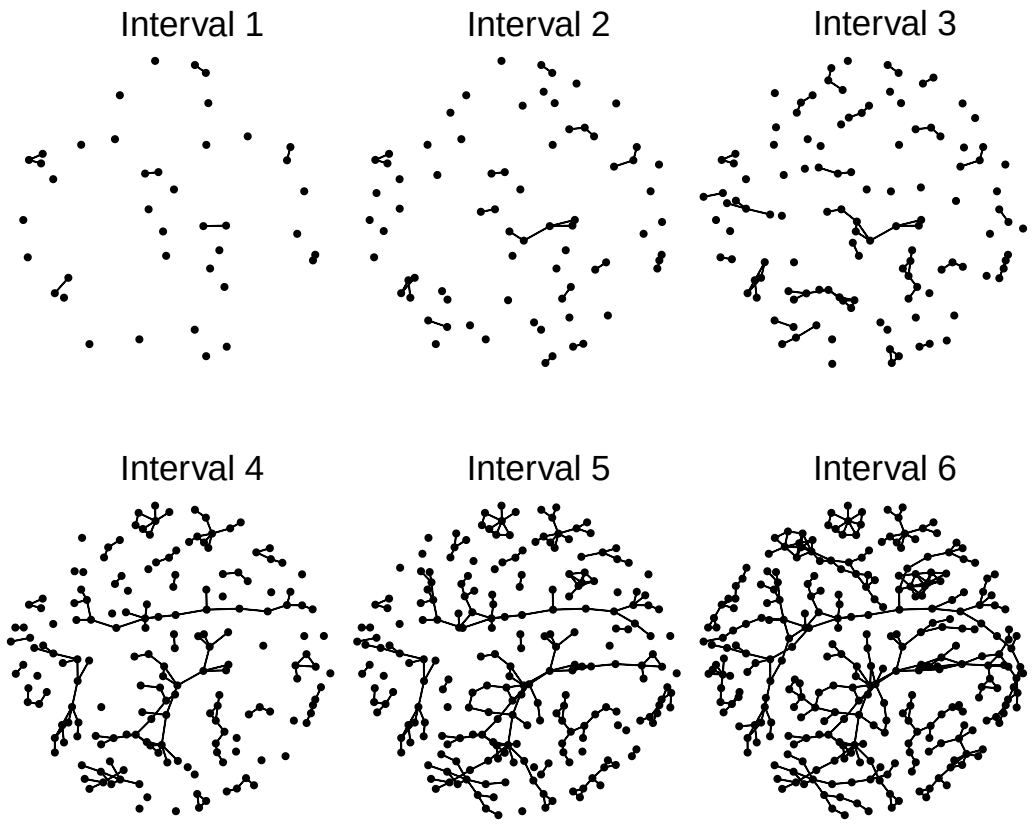

Fig. 5. Growth of the 10 largest connected components of monitored users over 30 days. Since new users are only monitored upon a change in their cheater flag status, we are able to get a high level view of the contagion process. As new users are banned, they form clusters with other previously banned cheaters.

changes in the flag by collecting friends lists for the next 10 days. The analysis component of the architecture allowed for new insight into the contagion process governing the spread of unethical behavior and the social penalty where newly exposed cheaters suffer a loss of friends in contrast to a control group of non-cheaters who gain friends.

One area not discussed in this paper is overall system performance. Although our implementation is able to process over 9 million inserts (and 10s of millions of reads) per day, the choice of data store has an impact. For example, MongoDB lacks partial indexes which could be leveraged to speed up queries of the "current" state of a user.

Finally, although the codebase was designed to meet our domain specific goals, the architecture lends itself to small units of easily understood code, and the abstractions we developed are usable in a general context. Thus, we are working towards a publicly available code release.

Acknowledgements The US National Science Foundation supported this research under grants CNS 0952420 and CNS 0831785.

REFERENCES

[1] W. Willinger, R. Rejaie, M. Torkjazi, M. Valafar, and M. Maggioni, "Research on online social networks: time to face the real challenges," *SIGMETRICS Performance Evaluation Review*, vol. 37, no. 3, Jan. 2010.

[2] A. Iamnitchi, J. Blackburn, and N. Kourtellis, "The social hourglass: An infrastructure for socially aware applications and services," *IEEE Internet Computing*, vol. 16, pp. 13–23, 2012.

[3] S. H. Lee, P.-J. Kim, and H. Jeong, "Statistical properties of sampled networks," *Phys. Rev. E*, vol. 73, p. 016102, Jan 2006.

[4] J. Blackburn, R. Simha, N. Kourtellis, X. Zuo, M. Ripeanu, J. Skvoretz, and A. Iamnitchi, "Branded with a scarlet "c": cheaters in a gaming social network," in *Proceedings of the 21st international conference on World Wide Web*, ser. WWW '12, 2012, pp. 81–90.

[5] B. Wellman, *Structural Analysis: From Method and Metaphor to Theory and Substance*. Cambridge University Press, 1988.

[6] N. Ruflin, H. Burkhart, and S. Rizzotti, "Social-data storage-systems," in *Databases and Social Networks*, 2011, pp. 7–12.

[7] S. Carrell, F. Malmstrom, and J. West, "Peer effects in academic cheating," *Journal of human resources*, vol. 43, no. 1, pp. 173–207, 2008.

[8] B. W. Kulik, M. J. O'Fallon, and M. S. Salimath, "Do Competitive Environments Lead to the Rise and Spread of Unethical Behavior? Parallels from Enron," *Journal of Business Ethics*, vol. 83, no. 4, pp. 703–723, Jan. 2008.

[9] F. Gino, S. Ayal, and D. Ariely, "Contagion and Differentiation in Unethical Behavior: The Effect of One Bad Apple on the Barrel," *Psychological Science*, vol. 20, no. 3, pp. 393–398, Mar. 2009.

[10] D. H. Chau, S. Pandit, S. Wang, and C. Faloutsos, "Parallel crawling for online social networks," in *Proceedings of the 16th international conference on World Wide Web*, 2007, pp. 1283–1284.

[11] D. Souravlias, G. Koloniari, and E. Pitoura, "Intersocialdb: An infrastructure for managing social data," in *Intersocial Workshop on Online Social Networks: Challenges and Perspectives*, 2012.