Poster: Distributed Streaming Reconstruction of Information Diffusion

Peter M. Fischer Io Taxidou Bernhard Lutz Michael Huber
{peter.fischer,taxidou,lutzb,huberm}@cs.uni-freiburg.de
University of Freiburg, Germany

ABSTRACT

Recent advances in social media have triggered a massive engagement of user population: a large part of people's lives has shifted to social media platforms and real events are reported while they are happening (e.g. in Twitter). As a result, such platforms have become an important source of information, being used by professionals as well, e.g. journalists, for fast access to news and events. Social media maintain an underlying network of social connections over which such information propagates. Information diffusion in social media has attracted attention, by analyzing how information is propagated from user to user and who is influenced by whom. Given the scale and speed of such information, systems that can keep up with such fast rates are required. In this poster, we present a system for real time reconstruction of information diffusion that encompass the challenges of analyzing fast data streams combined with large social graphs.

CCS Concepts

•Information systems → Data streams; Stream management; •Theory of computation → Streaming, sublinear and near linear time algorithms; Distributed algorithms; •Human-centered computing → Social network analysis;

Keywords

Information Diffusion; Realtime social media analysis

1. INTRODUCTION

Modern social media like Twitter or Facebook encompass a significant and growing share of the population, which is actively using it to exchange messages. This has a profound effect on the way news, events and all kind of information are spreading in terms of frequency, reach and speed. Given its broad coverage of the world as well as its fast reaction times, social media acts as a powerful "social sensor". Driven by this relevance, there is significant interest in performing fast, scalable and thorough analyses. Particularly, assessing the relevance and trustworthiness of a piece of information in a timely way is a key challenge.

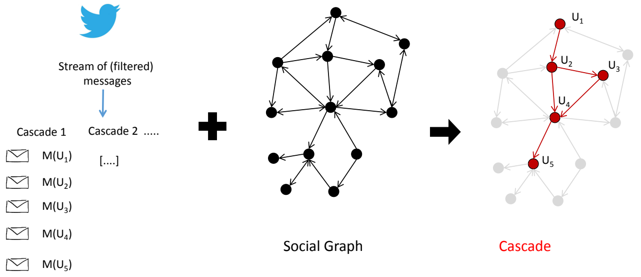

Figure 1: Streaming Influence Inference

Consider the user case of an online journalist. In order to take advantage of the rapid update rates, the huge amount of information and the coverage of events while they are happening, journalists need fast and accurate information diffusion analysis: 1) understanding how information reaches them, 2) what is their own impact after publishing, 3) detecting the full diffusion process and assess the relevance of trends, events and moods.

In order to tackle such a use case, systems that can derive such influence in real-time are needed. Information in social media arrives in fast rates, reaching virality (millions of messages) in short time and affecting a large part of the population. In order to keep up with the stream of messages, incremental and distributed computations are required. Complementary, given the need to trace information diffusion and identify who is connected with whom, processing the social graph in a fast way becomes a necessity. Considering the current sizes of social graphs (320M active users in Twitter), partitioning and distribution need to be performed in such a way that quality of results is not affected.

2. RECONSTRUCTING CASCADES

Although there is related work on how to model [3], reconstruct [1] and analyze the properties [11, 5] of information diffusion in term of "cascades" (temporal graphs with users as nodes and the spread/influence as edges), there is almost no research on how to perform such analyses on real-time in the presence of both very fast streams and huge social graphs.

In previous work [9], we have developed a single-site, incremental algorithm that can reconstruct information cascades containing retweets, i.e., explicitly forwarded short messages on Twitter. Extended versions of this algorithm are presented here to outline the challenges of reconstruction and possible improvements.

The algorithms handle retweets, since they constitute a common vector for information diffusion. Twitter, however, includes only information on the original message of a retweet, but not on the intermediate steps or the full cascade. Since it makes only minimal

DEBS '16 June 20-24, 2016, Irvine, CA, USA

ACM ISBN 978-1-4503-4021-2/16/06.
DOI: http://dx.doi.org/10.1145/2933267.2933294

assumptions on the model and data - such as order of exposure and social graph connectivity as carrier, the algorithm is also suited to reconstruct other types of diffusion, e.g. hashtags where the grouping of messages is much weaker. The idea core idea (as shown in Figure 1) is to correlate the stream of messages (and the users that emitted such messages) with the underlying social graph. Influence edges -coloured in red over the social graph- are derived when a user emits a message and is connected via the social graph to users who previously also emitted this content. In Figure 1 we see that when user U_4 emits a message two of his connections U_2 and U_3 have emitted the same message before, as a result we generate two influence edges that connect U_4 with U_2 and U_3.

A naive implementation of this idea would have cubic complexity: For each newly arriving message M_{n+1} we need to check if the user U_{n+1} who created it is connected to any of the users U_1 to U_N who created the previously arriving messages M_1 to M_N, where either the user has K connections or each of these users has J connections. Within the scope of this work we only consider direct connections, but for future research multiple hops may be relevant in order to compensate for missing information. Similar to join processing, we can now exploit indexing and domain knowledge: the order among the prefix or the connections may not matter, so we can treat them as set and get efficient containment checks. This leads to two algorithm variants that are shown in Figure 2): The first variant treats the follower connections of U_1 to U_N as sets and iterates over the prefix of messages seen so far (M_1 to M_N) to check if U_{n+1} is contained in any (or several) of these sets. The second treats the users in the prefix (U_1, ..., U_N) as a set and iterates over the friend connections of U_{n+1} to check if any of them is contained in the set. Clearly, the former works better for small cascades with heavily popular users, while the latter works best with large cascades of not-so-well-connected users. Variants of these algorithm can switch between these iteration types according to the relative size of each of these sets and the iteration targets, since both the cascade sizes and the connection set sizes are heavily skewed in practice. This computation provides incremental results (as each newly arriving message can be handled individually) at high speeds. In [9] we used prefix iteration and observed that the complete reconstruction of all influence paths in a cascade with around 10K messages is completed in less than 5 seconds.

Problem Statement and Challenges

Since we want to construct cascades on real time on all possible users we need to address the data scale and rapid rates. In periods of heavy traffic, hundreds of thousands of cascades are taking place in parallel, consisting of up to hundred of thousands or millions of retweets. Complementary, the social graph consists of hundreds of millions of users with dozens of billions of edges; clearly querying such a graph for real-time computations creates a considerable overhead, as its raw size by far exceeds the RAM available to commodity systems.

In turn, the main bottleneck of these algorithms is the size of iterations and the access latency to the sets. For the latter, we observed that even in the centralized algorithm it is massively affected by increased access latency. Switching from a hash- based approach to a sorted-list-based approach (in order to save space) gave slowdowns by more than an order of magnitude. In turn, a first attempt of distributing only the data by utilizing a request-response based approach to the social graph adjacency lists led to a slowdown by three orders of magnitude.

According to our observations, when computing the influences edges, we encounter very low selectivity in the connection sets: the average degree of a node in the cascade is slightly above one, yet there can be massive skew (such as the outdegree of a very popular user). The repeated access to connection sets at the beginning of a prefix might lend itself to bulk transfer of the entire set, but this skew might also be very costly. In other words, we need to repetitively probe possibly large sets, while the return set is very small. Due to the number of accesses a typical request/response approach will incur significant additional latency, while transferring full adjacency lists can be rarely amortized.

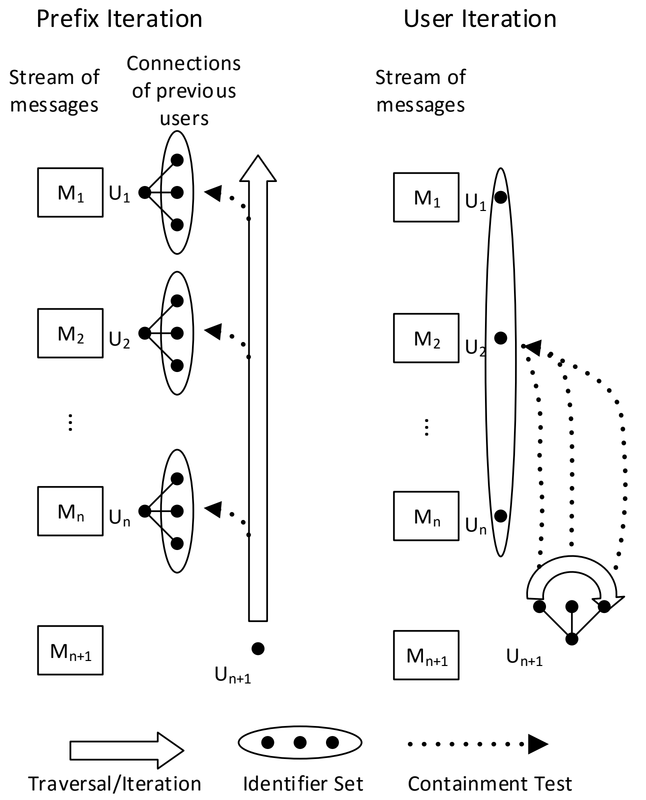

Figure 2: Reconstruction with Prefix and User Iteration

There is not much work that combines classical streams with large graphs; typical graph processing systems like GraphLab [7] are efficient in iterative graph computations, but do not cater for streams. Complementary, such a problem is not a classical stream processing problem where a single pass over the data is desired: it is an iterative stream processing problem where we need repetitive access to the stream prefix (set of messages or user connections). The only known approach is Naiad [8] which allows for iterative computation over data streams using timed dataflows. As such, Naiad provides strong coordination means, something that is one of the next issues to tackle. It does, To the best of our knowledge, it does not provide the fine-grained control over algorithm and state distribution we are trying to achieve here.

3. DISTRIBUTED PROCESSING

Given these challenges, we aim for a distributed approach that minimizes the access latency on the social graph, and keeps the computational cost and number of messages to transfer low.

We follow a number of design ideas to express this distribution efficiently: The first is to partition the set of adjacency lists in a non-overlapping way to achieve a good scalability in terms of the massive graph sizes. The second is to perform all iterative accesses to an adjacency list and checking of set containment locally to avoid

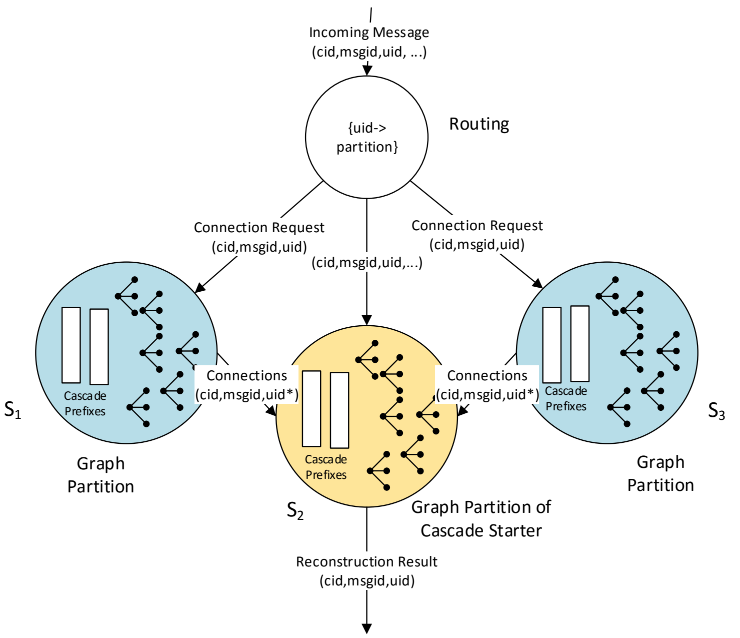

Figure 3: Distribution of Graph Data and Processing

the latency penalties of such accesses. Taken together, these two ideas mean that we may need to distribute the iteration over the messages over multiple nodes. To reduce the latency and amount of data to transfer when processing different messages at different nodes, we take in turn three steps: a) We collect all results (and perform the related computation) at the site of the cascade starter, minimizing distributed operations altogether. b) We combine the low selectivity of lookups with an adapted partitioning strategy to minimize the amount of result data and the time to transfer. c) We utilize stateful processing and parallel sending to reduce the size and time of messages' requests at the different nodes. Lastly, we perform parallel processing to reduce the overall response times.

All theses ideas translate into the following architecture that builds on the infrastructure we developed for the centralized algorithm, as shown in Figure 3. We assign the adjacency lists of the social graph to the processing nodes according to the partitioning; we centrally retain information on the locations of this assignment. Utilizing the cascade in Figure 1, let us (as an example) assign user U_2 to S_1, U_1 as well U_5 to S_2 $U3$ as well as U_4 to S_3.

For newly arriving information in the cascade, we send a triple of identifiers (cascade, message, user) in parallel to the relevant partitions to perform partial computation. This partial computation provides the part of the connections for this user that can be derived at this partition. To gain a full reconstruction of the cascade (containing all edges in the correct order), the partial results are combined at the partition containing the user who started the cascade. This decision is driven by the insight (also confirmed by our previous work [9]) that such users often have significant influence (and thus many outgoing edges) in the cascade.

The partial prefix information and the computation at each site as well as the information to send to a site depend on the iteration of the reconstruction algorithm: For prefix iteration, each partition will be used to iterate over the intersection of the users in the cascade so far and the user set present at the site. Using the running example cascade and the allocation described before, the connections of message $M(U_5)$ will have to be checked against site S_1,S_2 and S_3, since each of these sites contains users from the prefix of the cascade (U_2, U_1 and U_3/U_4, respectively). In order to avoid sending the (growing) prefix to the affected sites over and over again when probing, we keep the partial prefix (intersection) for each active cascade at each site (e.g., U_3/U_4 at S_3). Generating this prefix can be piggy-backed with connection checking: when we probe for the connectedness of a user, this user is added to the prefix at the site where its own connections reside. This way, messages of (small) constant size are sent to the subset of set of sites that contain users in the prefix of the cascade. For user iteration, all potential connections of a new message can be found at a single site, yet the full set of users active in the cascade needs to be accessed (see Figure 3, right). We therefore collect the complete prefix at each site involved in the cascade, and transfer the existing prefix in bulk when a so -far inactive- site is accessed.

In both cases, each partition keeps a partial state of the computation so that the cascade can be processed in parallel. Overall, both approaches will converge to sending all messages to all active sites. Since these requests are performed in parallel to regular processing, they do not significantly affect the overall latency. Since the selectivity of these requests is very low, the response counts (and sizes) are significantly lower, providing even less of a bottleneck. We currently synchronize the reassembly (including reordering and result generation) at the end of a cascade, but we are working on more fine-grained synchronization which may also tie into batching. Investigating the best sizes for such batching as well as understanding the tradeoff between minimizing cost and achieving load balancing are some of the challenges we plan to investigate. We also need to understand for how long such a distribution stays stable and to which extent adaptive, fine-grained migration of connections might be needed.

Despite these significant optimizations in processing the cost, a suitable partitioning will still make a significant difference in the overall performance. Our problem differs from both the traditional graph partitioning as well as as social graph partitioning for high variance in out-degree [2]: 1) The graph partition needs to exploit skew to ensure that interactions of users remain local. 2) The size of a partition is not determined by the number of nodes or intra-partition edges, but by the overall edges outgoing from users.

We aim to solve this problem using a key observation and derived hypothesis to perform the partitioning: Past interactions [4] (also past influence) are an important factor to assess the strength of links in the social graph: Users who interacted in the past are more likely to interact in the future. To exploit this idea, we have partitioned the interaction graph of users [10] which reveals who interacted with whom, and use this partitioning to allocate the social graph. In our workload, these graphs are actually a byproduct of computing influence, so we can easily collect the results and learn over time.

4. PRELIMINARY EVALUATION

So far, we have performed a set of preliminary experiments, determining the cost increase of a single-site reconstruction. For a deeper understanding of the cost increase and the efficiency of our partitioning, we also investigate how many remote requests are being issued and how many remote responses are being generated. Since the exact time needed for requests and responses is hard to be measured reliably, we compare the total time of performing the reconstructing at a single site (with just the subset of the social graph needed) against the time to perform distributed reconstruction.

The evaluation was performed on selected cascades from the dataset we used for [9] that represent a typical diffusion workload. We collected messages from the Twitter Streaming API from August 3rd to September 24th 2012, using the filter terms "Olympics" and "London2012". In total the data set contains almost 11 million tweets, in particular 1.1 million separate retweet cascades. The largest cascade has more than 60K retweets, around 150 have more than 1K retweets, approximately 5000 cascades have more 100 retweets and around 45000 cascades contain 10 or more retweets. We also used the REST API to retrieve the social graph connections (follower information) for those users present in the cascades. Overall, the social graph we are using contains around 4.5 million follower lists with more than 4.8 billion edges and requires around 45GB of storage on disk.

Out of this dataset, we selected a number of cascades with different properties: a) individual cascades with 60K, 29K, 15K, 7K and 3K messages. b) several collections of cascades with around 60K messages in total: i) 500 cascades at around 120msg ii) 100 cascades at around 600msg iii) 10 cascades at around 6000msg.

The experiments were performed on a Cluster of 9 machines, each with an Xeon E5-2420 (1,9 Ghz, 6 cores) and 32 GB of RAM. This cluster ran Storm 0.9.6 on top of OpenJDK 7 and Ubuntu Linux 12.04 LTS (each in the 64 bit version). The Storm topology was configured to use 1,2,4 or 8 nodes for graph partition and reconstruction sites and one additional node to perform the event stream loading, routing etc. All involved bolts were pinned to specific machines using a custom scheduler. The social graph fragments at each site are loaded when deploying the topology, and run a single bolt with around 30 GB JVM heap at each of the reconstruction sites. The allocation was performed using the interaction network of this data (utilizing knowledge on the link strength) and partitioning it using the METIS [6] framework. Given that the dataset only contains follower information, we performed all evaluations using prefix iteration.

Comparing individual cascades (a) clearly shows the overall usefulness of our approach. With one exception, the runtime overhead was less than 5 percent of the total time when using a single site. In the cascade with 15K messages, the main influencer is not the starter of the cascade, but a more popular “downstream” user. In this case, the reconstruction time increases less than a factor of two - which is still quite usable in practice. The number of remote responses gives a clear indication on these effects: for all cascades except the one with 15K messages, the number of edges that are delivered remotely is between 1.5 and 12 percent of the cascade size. For the outlier cascade with 15K, this number even exceeds the cascade, as a) most users are not in the starter’s partition and b) many nodes have multiple ingoing influence edges. Expressing the computation in a stateful manner saves a significant amount of time and message size over repeated prefix requests, cutting the overhead between 20 and 90 percent.

When running sets of cascades in an interleaved manner, we compared the total completion times with the sum of the individual reconstruction times to understand the competing effects of parallelism and resource contention. Generally speaking, we see a speedup for all of these cascades when comparing against a single-site reconstruction. This effect becomes more pronounced with fewer and larger cascades where the total time is significantly less that the sum of all individual reconstruction times.

Overall, the results show that the overheads of distributions are significantly lower than those of approaches based on request/response patterns over distributed storage or stateless computation. Future evaluations will include a comparison against a direct implementation in Naiad and further investigation on balancing computation and communication.

5. CONCLUSION

Analyzing information diffusion provides important insights into the relevance and trustworthiness of social media. Yet, very little of the existing work has tackled the challenges of performing information diffusion reconstruction at the scale and speed of contemporary social media. In this poster, we present a system and distributed algorithm to combine high-rate message streams and massive social graphs to infer such influence in realtime. Both social graph information and the computation are distributed so that communication cost is minimized and parallel computation is increased. The preliminary performance results reveal that the system shows great promise to address those challenges.

6. REFERENCES

[1] P. Cogan et al. Reconstruction and Analysis of Twitter Conversation Graphs. In *HotSocial ’12*, 2012.

[2] J. E. Gonzalez et al. PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs. In *OSDI*, volume 12, 2012.

[3] A. Guille et al. Information Diffusion in Online Social Networks: A Survey. *SIGMOD Record*, 42(2), 2013.

[4] B. Huberman et al. Social Networks that Matter: Twitter Under the Microscope. *First Monday*, 14(1), 2008.

[5] C. Hui et al. Information Cascades in Social Media in Response to a Crisis: A Preliminary Model and a Case Study. In *WWW (Companion Volume)*, 2012.

[6] G. Karypis and V. Kumar. A Fast and Highly Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM*, 20(1):359–392, 1999.

[7] Y. Low et al. Distributed GraphLab: A Framework for Machine Learning and Data Mining in the Cloud. *VLDB Endowment*, 5(8), 2012.

[8] D. G. Murray et al. Naiad: A Timely Dataflow System. In *SOSP*, 2013.

[9] I. Taxidou and P. M. Fischer. Online Analysis of Information Diffusion in Twitter. In *WWW (Companion Volume)*, 2014.

[10] C. Wilson et al. Beyond Social Graphs: User Interactions in Online Social Networks and Their Implications. *TWEB*, 6(4), 2012.

[11] Z. Zhou et al. Information Resonance on Twitter: Watching Iran. In *Social Media Analytics*, 2010.