Real-time Analytics for Fast Evolving Social Graphs

Charith Wickramaarachchi[1], Alok Kumbhare[1], Marc Frincu[2], Charalampos Chelmis[2], and Viktor K. Prasanna[2]
[1]*Dept. of Computer Science, University of Southern California, Los Angeles, California 90089*
[2]*Dept. of Electrical Engineering, University of Southern California, Los Angeles, California 90089*
Email: kumbhare@usc.edu, cwickram@usc.edu, frincu@usc.edu, chelmis@usc.edu, prasanna@usc.edu

***Abstract*—Existing Big Data streams coming from social and other connected sensor networks exhibit intrinsic inter-dependency enabling unique challenges to scalable graph analytics. Data from these graphs is usually collected in different geographically located data servers making it suitable for distributed processing on clouds. While numerous solutions for large scale static graph analysis have been proposed, addressing in real-time the dynamics of social interactions requires novel approaches that leverage incremental stream processing and graph analytics on elastic clouds.**

We propose a scalable solution based on our stream processing engine, Floe, on top of which we perform real-time data processing and graph updates to enable low latency graph analytics on large evolving social networks. We demonstrate the platform on a large Twitter data set by performing several fast graph and non-graph analytics to extract in real-time the top k influential nodes, with different metrics, during key events such as the US NFL playoffs. This information allows advertisers to maximize their exposure to the public by always targeting the continuously changing set of most influential nodes. Its applicability spans multiple domains including surveillance, counter-terrorism, or disease spread monitoring. The evaluation will be performed on a combination our local cluster of 16 eight-core nodes running Eucalyptus fabric and 100s of virtual machines on the Amazon AWS public cloud. We will showcase the low latency in detecting changes in the graph under variable data streams, and also the efficiency of the platform to utilize resources and to elastically scale to meet demand.

***Keywords*-real-time graph analytics; social networks; cloud computing; stream processing**

I. Overview and Background

Online social networks are complex environments characterized by fast changes in the topology and dynamic interactions between members. The advent of social services such as Facebook and Twitter has catalyzed their evolution and has opened up numerous interdisciplinary research opportunities for analyzing their structure as social entities and establishing theories for the observed patterns. Data sciences play an important role in this process by enabling insight on the social networks through data analysis techniques. While promising, the efficiency of applying such techniques for low latency real-time analysis is limited by the growing complexity and dynamics found in social networks. To give an example, Twitter has an average of 6,000 tweets per second however during peak, rates up to 143,200 tweets per second have been observed. In Facebook's case we notice an average of 41,000 posts per second or about 2.4Mb of data each second. Processing this huge amount of fast streaming data to extract useful knowledge in real-time is challenging and requires besides efficient graph updates, scalable methods for performing incremental analytics in order to reduce the complexity of the data-driven algorithms.

Existing graph processing methods have focused either on large shared memory approaches where the graph is streamed and processed in-memory [6], or on batch processing techniques for distributed computing where periodic graph snapshots are taken and processed independently [9]. While these techniques are efficient for some classes of evolving graphs where data is either centrally stored or their structure is slow changing, present day online social networks where data is gathered in geographically distributed data centers (i.e., on the cloud) and which exhibit fast topology changes. At the other end of the spectrum we find streaming engines such as Apache Storm or our Floe streaming engine [8] which are capable of processing fast streaming data at low latency but lack graph abstractions that can leverage the evolving inter-dependencies found in social networks. Incremental algorithms for graph based analytics have been previously proposed [5] however they have yet to be integrated into streaming platforms.

As part of our research activities, our group is actively involved in scalable fault-tolerant stream processing engines tailored for clouds and scalable incremental analytics on fast evolving graphs. Specifically we address: (1) the adaptability, fault-tolerance, and scalability required by the stream processing pipeline to continuously ingest high-velocity, variable data streams; (2) the graph abstractions required to enable incremental graph analytics on distributed memory systems; and (3) the scalability of specific data-driven graph oriented analytics. Our integrated platform, addressing the above problems, is built on top of our streaming engine Floe which allows fault-tolerant, elastic, and stateful stream processing at higher peak throughput as compared to Storm [1]. Incremental graph algorithms are implemented on top of it to allow real-time graph analytics.

The focus of this challenge is on demonstrating the scalability and low latency processing capabilities of our platform for real-time analysis of fast evolving graphs. We

target a specific use case involving the detection of evolving key nodes in social networks. The motivation behind this use case is driven by the countless examples of practical applications it enables. These range from advertising and surveillance to disease spread monitoring. Taking for instance a simple advertising scenario during a major event such as the NFL playoffs, an ad company could try to maximize its reach by targeting key nodes in the social network.

The rest of the paper is structured as follows: Section II gives the details of the social network application; Section III describes and analyzes the novel features of the scalable architecture for graph enabled streaming analytics; and finally Section IV illustrates the demonstration scenario and performance metrics for the challenge.

II. Real-time Social Network Analysis for Uncovering Key Nodes

The goal of our real-time social network analysis is to find the set of key nodes (or people) and how it evolves over time. Here we determine a *key* node based on various metrics such as out edge degree (e.g. number of re-tweets), cluster coefficients (i.e. how connected is a node's neighborhood), and influence seed nodes (i.e. set of nodes that should be targeted to start a information cascade). The goal of the application is twofold: (1) identify popular people over time in real-time; and (2) identify bots, under the same considerations, which look for specific keywords and automatically re-tweet the messages (see §III-C).

As seen next, our application deals with several key problems such as a fluctuating data streams and a constantly changing set of key nodes. The application is divided into 3 parts: (1) information integration, (2) data analytics, and (3) information diffusion. Here we explain their characteristics:

A. Graph Updates based on Fast Stream Processing

Streaming data is by nature unstructured and any dependencies need to be extracted by considering its properties and by analyzing the specific use case. Complex applications built on top of social networks consider numerous sources combined together. This is made possible by the strong integration between various online social media, e.g., Facebook posts can be automatically tweeted and vice versa. Semantics help manage the processing of this complex data once it is extracted, normalized and semantically annotated by a stream processing engine such as Floe.

Based on semantics the graph can be updated by relying on latest information. Given the numerous views such complex data has, deciding the correct representation for the application at hand is a crucial step. Taking as an example the Twitter data, it can be modeled as a graph where nodes represent users and edges their interactions based on tweets, re-tweets, or mentions. Furthermore directed graphs can model outgoing edges as re-tweets or mentions, and incoming edges as re-tweets of other users' tweets.

The stream processing engine needs to adapt to several forms of dynamism. First, the stream rate can vary greatly based on time-of-day or event and can scale up by a factor of 100 in a few seconds. Second, we need to consider that at any given point the number of active users can vary significantly impacting specific regions in the graph that may or may not be geographically collocated. This leads to a fine choreography between fast elasticity and load balance that needs to be carefully planned if analytics are to be performed in real-time. Finally, it needs to cope with potential system failures which may render the real-time processing useless.

B. Incremental Graph Analytics Algorithms for Identifying Key Nodes

Identifying key nodes in evolving graph requires an efficient method that is robust and adaptive to the velocity and variations in data streams. Various techniques based on centrality measures [4] and more recently the tipping model [7] exist. Irrespective of the method, traditional approaches consider static graphs or periodic snapshots on which they run the entire algorithm and recompute the set from scratch. In dynamically evolving social graphs this approach is intractable, especially for real-time operations. To enable scalable robust solutions we adopt an incremental analytics approach where a new solution is incrementally computed based on the previous one and graph updates that arrived since the last computation.

We rely on a recently proposed algorithm based on shell decomposition [7] as it was demonstrated to outperform the classic centrality measures and show robustness against the removal of high-degree nodes. This algorithm allows us to determine the smallest possible set of individuals (seed set) such that, if initially activated, the entire population will eventually become activated, adopting the new property, or in our case re-tweeting the ad. The algorithm is highly dependent on updating the graph hence its efficiency depends on how fast the semantically annotated graph updates are applied to the existing graph. These updates can take place continuously as they arrive or in batches of fixed length or taken at fixed intervals. During low streaming rates (hundreds per second) a batch approach may be more suited whereas during high streaming rates (thousands or tens of thousands per second) a continuously processing would be desired. The size of the batch or the interval at which it is created depend on the processing capabilities of the streaming engine. In our case we will showcase the advantages of an elastic architecture for scaling to meet the demands of any real-time graph structured stream by employing incremental continuous analytics. Finally, we develop an incremental algorithm to identify the evolving set of triangles in the graph and hence calculate the cluster coefficient to identify evolving cliques in the network in order to mark potential bots which could be used to speed up the propagation of our ads.

The mapping of these algorithms on our proposed scalable solution for real-time analytics on fast changing social graphs will be detailed in Section III-C.

C. Visualization Portal

The application is intended for two types of information consumers, namely system operators, and data analysts. Operators visualize real-time information on the key nodes to decide on operational changes such targeting newly introduced players by following them or monitoring they activity. Analysts evaluate the effectiveness of different interaction strategies, or the evolution of the key players during events or over a period of time.

A web based visualization portal serves as a central service for this information diffusion. It offers a view of the social network graph overlaid with the key nodes at the present time. The view offers the possibility to scroll back in time and to visualize the evolution of the set. Given the real-time analytics we perform the view is constantly updated with the latest information on the key nodes. The portal also offers a view on some system specific metrics such as set update latency, current stream rate, computational resources' load. The portal relies on historical already processed data stored in a repository and real-time data from the data stream.

III. Scalable Architecture for Graph Enabled Streaming Analytics

To enable real-time graph processing we augmented our initial Floe scalable distributed stream processing framework with support for run-time elasticity, load-balancing, and low latency fault-tolerance. On top of this improved version we added an incremental graph processing framework for low latency analytics on evolving graphs. The former is used to pre-process high-velocity raw data streams, enable information integration, and generate relevant graph update events. The latter is used to consume graph update events and execute continuous graph analytics for tracking high influence nodes in the evolving graph.

A. Elastic Stateful Stream Processing using Floe

Floe enables applications requiring continuous stream processing and analytics by following the familiar directed acyclic graph (DAG) [1], [10] model. In the DAG abstract dataflow, the nodes in the graph represent the processing elements (PEs) while the edges represent the direction and flow of data between these PEs. Floe applications are deployed on a distributed cluster and offer scalable operations through message partitioning, process replication and data parallel operations. Floe's distinguishing features that separate it from other solutions are: (1) support for runtime elasticity, both within (scale up/down) and across virtual machines (scale in/out); (2) efficient state management and seamless state migrations to support elasticity for both stateless and stateful processing elements; and (3) integrated load-balancing and fault-tolerance with low-latency (sub-second) fault recovery.

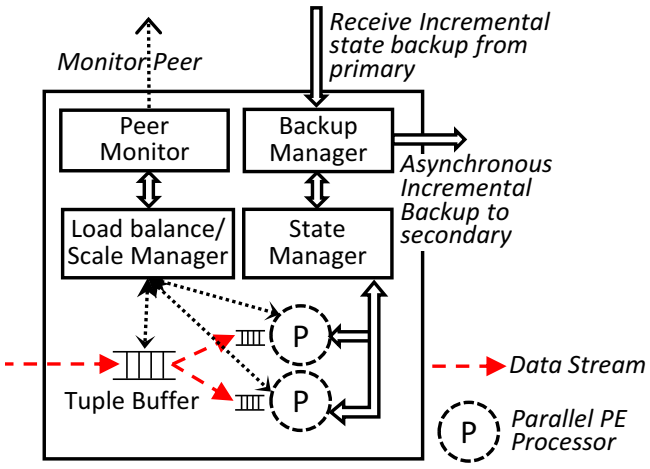

Figure 1. Floe container components.

The above features are achieved through an integrated framework that utilizes consistent hashing to form a peer-ring which is used both for state and message partitioning among the peers (cf. Figure 1). Each node consists of a peer monitor, a state and message backup manager, as well as scale manager as shown in figure 1. The state and backup manager is responsible for asynchronous incremental state checkpointing and backup of the state as well as unprocessed messages on the neighboring peers, while the scale manager is responsible for monitoring the load (e.g., buffer length, memory usage, CPU usage) on the node and initiate either load balance or scaling in/out operations based on the negotiations with its neighbor on the peer ring. The peer ring coupled with incremental backup and continuous monitoring allows the system to elastically scale the system with very low latency. It also supports efficient load balancing and low latency fault-recovery. Figure 2 shows the latency characteristics for the load-balancing operation (i.e., to shed some of the load to the neighbor on the peer ring), which involves both message replay and state migration. We observe low latency (sub-second to few seconds) for such operation for a range of data rate and checkpointing period. We also observe similar characteristics for scaling in/out and fault-recovery operations, however, the plots are omitted for brevity.

B. Evolving Graph Analytics using the Proposed Graph Framework

We developed an incremental graph framework on top of Floe to support efficient storage and incremental processing of evolving graphs. The framework leverages the stateful processing elements supported by Floe to distribute the evolving graph across the cluster. This in turn provides

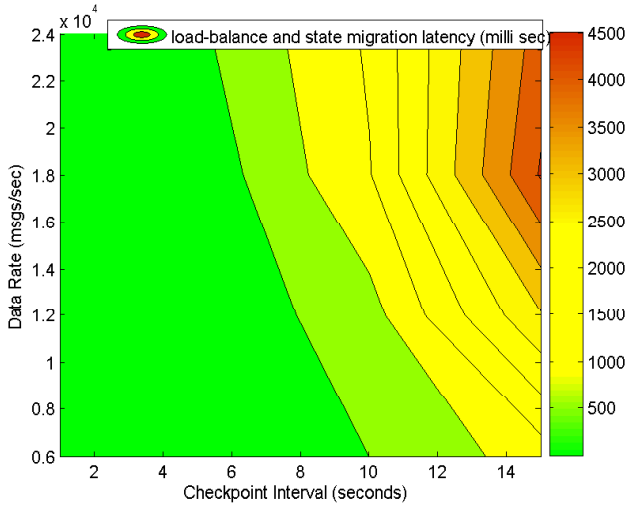

Figure 2. Backup node latency to restore state and replay tuples.

runtime elasticity and dynamic load-balancing required for high velocity graph analytics on evolving graph. To the best of our knowledge the proposed framework built on top of a streaming engine is the first system to enable fast information integration and incremental graph algorithms at large scale with support for elasticity and fault-tolerance.

Figure 3 shows various components of the incremental graph framework built on top of Floe. The *Graph Update Event Generator* component is a Floe application that processes the incoming raw data streams and generates graph update events (e.g., add/remove vertex, add/remove edge, update edge value/weight, and update vertex value). These graph updates are consumed by the *Dynamic Graph Partitioning* component which uses online partitioning algorithms that balances the processing cost across the cluster as well as minimizes the overall communication cost by reducing the edge cuts across the nodes. In addition, it also has to minimize the state and partition migration at runtime. Note that these requirements exhibit potentially conflicting decisions and the partitioning algorithms use a optimization function that maximizes the overall throughput.

The proposed framework takes graph updates as an input and performs analytics in an incremental manner. Updates are taken as an input in batches and processed. At the end of the processing updates analytics (if any) are pushed out so they can be used for further analytics or visualization. The objective is to provide low latency analytics which is scalable with the graph size and the graph update rates.

The user is provided with a programming abstraction that takes graph updates as an input and enables analytics implementation. APIs are provided to access the graph structure and perform operations, and also to maintain the application state required for handling the distributed incremental graph analytics. This way, the framework handles the distributed state management transparently to users. The graph processing programming abstraction also hides the communication between workers. Users are provided with a simple message passing API to communicate with other workers. Mapping details about neighbor vertices that owed by other workers are maintained within each worker. All graph updates and inter worker messaging is done using the data channels of Floe.

The proposed framework initially assigns the graph updates to set of workers using fast local heuristics (e.g., hash based). Workers then periodically re-partition the graph based on system characteristics to further improve the partitioning quality. Each worker is responsible for processing a graph partition which is stored as an adjacency list. Each partition is i responsible for a subset of vertices V_i such that $\bigcap_{\forall i} V_i = \emptyset$ and $\bigcup_{\forall i} V_i = V$. Here V represents the set of vertices in the graph and E the edges. To reduce the inter-worker communication overhead some of the data is replicated among workers. This approach is particularly useful for some algorithms such as the cluster coefficient presented in Section III-C.

We extend the Floe's auto-scaling algorithm to consider not only the load on the system (such as data rate, message queue length, CPU/memory usage), but also on graph and application specific metrics such as graph access patterns, edge cuts, and inter-node communication. Further, we extend the state management APIs to allow dynamic graph repartitioning and merging as a result of scaling in/out. These features allow the incremental graph analytic algorithms to transparently adapt to the variable data streams and data load and maintain the desired throughput.

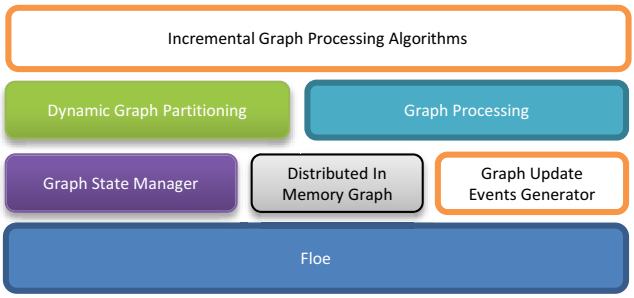

Figure 3. Architecture and components of the incremental graph framework.

C. Incremental Graph Algorithms

We developed series of graph algorithms to identify key nodes in social networks during key events such as the NFL playoffs. Our main goal is to identify key nodes in the networks that should be targeted for online marketing applications. Three main analytics have been identified: 1) high degree (in/out) nodes, 2) nodes with high cluster coefficient, and 3) seed sets nodes to initiate an information cascade.

High degree nodes in a social network can either represent popular/influential personal or bots. They can be differentiated based on the direction of communication with other nodes in the network. Identifying influential nodes can be useful for marketing or information propagation. Nodes with high cluster coefficients are normally automated bots in the network. Some of the bots scan keywords in information feeds and act as repeaters for these information. So bots can be used to get some level of visibility in advertising campaigns. Early adopters plays a major role when it comes to starting a information cascade. Identifying and maintaining the best set of early adopters to start propagating information is a critical in a competitive dynamic marketing environment to achieve information cascades.

High Degree Nodes. In interaction graphs such as social networks, nodes represent users and edges represent interaction between users. Out edges from a node repent initiation of a communication by a node. target node is refereed in that communication. (e.g., A re-tweets one of B's tweet means there is a edge from A to B).

In this case high out degree nodes represents the nodes that initiate most of the communication in the network and in most cases they are automated bots. In-degree nodes can be very popular people like celebrities. Bots in social networks normally look for some keywords and broad cast these messages to their networks. Identifying bots can be important for online marketing as they can be used to broadcast the message we want by appending the relevant keywords to the messages. Finding celebrities in a network is important since we can target them for advertising.

The high degree node finding algorithm uses the streaming map-reduce support provided by Floe. Initially, graph updates are extracted from social media streams. Then nodes with edges are grouped at reducers using key as the source. Those reducer outputs are forwarded to a different processing unit and where top high degree nodes are maintained for visualization. Reducer results are also written to the in-memory distributed graph asynchronously with updated degree statistics.

Cluster Coefficient. The cluster coefficient of a node in a graph quantifies how close its neighbors are to being a clique (complete graph). Generally nodes with high cluster coefficient are bots in the network. The cluster coefficient C_i for a vertex v_i is given by the number of links between the neighbors of a vertex v_i divided by the number of all possible links between neighbors.

In our online algorithm to find high cluster coefficient nodes is based on online triangle counting. We maintain number of triangles associated with each vertex and compute the cluster coefficient (CC) for each vertex using the triangles associated with each vertex. Cluster coefficient of a vertex can be calculated using following formula.

$$CC \text{ for vertex } i = \frac{2*T_i}{k_i(k_i-1)}$$

Where T_i represents the number of triangles associated with vertex i and k_i the degree of vertices.

We developed an online distributed memory triangle counting algorithm extending [3] using Floe. This algorithm emits the changes in number of triangles associated with each vertex if there is any changes in the current counts. Then cluster coefficient is recalculated for these vertices. Updated cluster coefficients are passed for processing and current list of high cluster coefficient nodes are updated if necessary.

Minimal the Seed Set. Identifying the minimal seed node set to start an information cascade is a important problem in viral marketing. These seed nodes can be used as early adopters to start a full cascade.

We developed an incremental algorithm based on the original algorithm presented in [7] and implement it in our framework by using its distributed message passing model. The static graph algorithm [7] is a communication heavy algorithm which needs heavy communication. The proposed incremental algorithm tries to perform local updates with minimal communication across workers to maintain the minimal seed set. Details of the distributed memory incremental algorithm is out of the scope of this paper.

D. Dynamic Graph Visualization

To visualize the real-time analytics we will rely on D3 [2], that enables data driven visualization. It supports abstract graph visualization with multi-level visualization that lets the user zoom in and out to show more or less details of the graph. Further, it supports advanced layering to visualize the graph in the context of the domain specific layers (e.g. twitter stream layered on top of geographical map). We extend the D3 visualization tool [2] to support visualization of evolving graphs that lets the user track various properties of the graph in real-time. We also overlay the incremental results obtained from the graph analytics that allows the user to track the algorithm's progress as well as partial results. Figure 4 shows a sample snapshot with key nodes identified for different connected communities in the South America overlayed on top of the geographical map.

IV. DEMONSTRATION

Our demonstration of the online graph analytics and applications (available at http://tsangpo.usc.edu/realtimegraph) will showcase the scalability and run time elasticity of the proposed incremental graph analytics framework built on top of an enhanced Floe engine. We will evaluate and demonstrate scalability with respect to the following metrics:

- Application Level Metrics:
  - Timeliness - i.e., time between the time that graph update enters a system and the time system finish performing for that update. This is a latency measure.

Figure 4. Visualization portal for Dynamic Graph and Incremental analytics.

- System Level Metrics:
  - Average queuing latency vs. data rate;
  - Average output throughput vs. data rate;
  - Scaling in/out latency (i.e. the amount of time spent during message, graph and state migrations, if required, during scaling operations);
  - Resource utilization and distribution across the cluster.

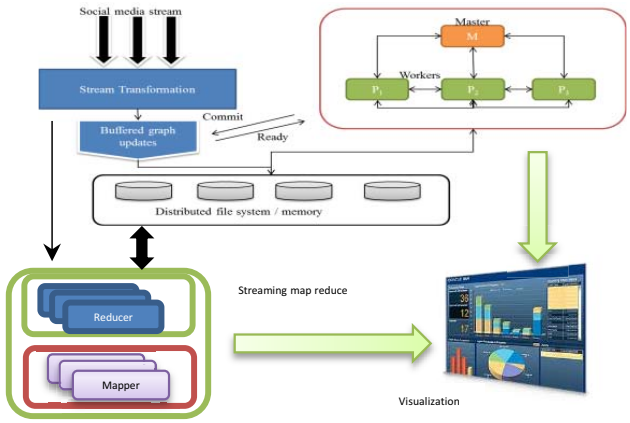

Figure 5. Demonstration Setup

Figure 5 shows the architecture of the application and setup. Twitter streams at full capacity (e.g., Firehose) will be injected into the system, which is then converted into graph update events such as vertex add/remove, edge add/remove, etc. The graph analytics algorithm then incrementally identify and update the key nodes in the graph based on various parameters such as geography, topics, hash tags etc. The graph along with the analytics is finally consumed by the visualization component that allows the user to interact with the graph and obtain relevant information.

The demo application will be run on a hybrid cloud, with a combination our local cluster called `Tsangpo`, which is running the Eucalyptus v2 cloud fabric and the Amazon AWS public cloud for additional resources. The Tsangpo cloud infrastructure located at USC provides seventeen nodes, each with 8-core AMD Opterons rated at 2 GHz each and 16 GB RAM, for a total of 136 available cores. Floe will scale out on this Cloud through on demand VM resource acquisition and further scale out to 100s of VM on Amazon in response to the system load. We will simulate variable data rates for the twitter stream and study the system response and scalability with respect to the metrics defined earlier.

ACKNOWLEDGMENT

This work was supported by the U.S. National Science Foundation under grant ACI-1339756.

REFERENCES

[1] Storm: Distributed and fault-tolerant realtime computation. http://storm.incubator.apache.org/. Accessed: 2014-06-30.

[2] D3: visual analytics for big data, Feb 2015.

[3] S. Arifuzzaman, M. Khan, and M. Marathe. Patric: A parallel algorithm for counting triangles in massive networks. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, pages 529–538. ACM, 2013.

[4] L. C. Freeman. Centrality in social networks conceptual clarification. *Social networks*, 1(3):215–239, 1979.

[5] A. Pavan, K. Tangwongsan, S. Tirthapura, and K.-L. Wu. Counting and sampling triangles from a graph stream. *Proc. VLDB Endow.*, 6(14):1870–1881, Sept. 2013.

[6] J. Riedy, D. A. Bader, and H. Meyerhenke. Scalable multi-threaded community detection in social networks. In *Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum*, pages 1619–1628, 2012.

[7] P. Shakarian, S. Eyre, and D. Paulo. A scalable heuristic for viral marketing under the tipping model. *Social Network Analysis and Mining*, 3(4):1225–1248, 2013.

[8] Y. Simmhan and A. G. Kumbhare. Floe: A continuous dataflow framework for dynamic cloud applications. *CoRR*, abs/1406.5977, 2014.

[9] Y. Simmhan, A. G. Kumbhare, C. Wickramaarachchi, S. Nagarkar, S. Ravi, C. S. Raghavendra, and V. K. Prasanna. Goffish: A sub-graph centric framework for large-scale graph analytics. In *Proceedings of the 20th International Conference on Parallel Processing*, EuroPar '14, page in print, 2014.

[10] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: cluster computing with working sets. In *USENIX conference on Hot topics in cloud computing*, 2010.