A Distributed Algorithm for the Efficient Computation of the Unified Model of Social Influence on Massive Datasets

Alex Popa *, Marc Frincu*†, Charalampos Chelmis‡
*West University of Timisoara
Faculty of Mathematics and Computer Science
Blvd. Vasile Parvan No. 4, 300223, Timisoara, Romania
Email: {alex.popa87, marc.frincu}@e-uvt.ro
†e-Austria Research Institute Romania
Blvd. Vasile Parvan No. 4 Room 045B, 300223, Timisoara, Romania
‡University at Albany
College of Engineering and Applied Sciences
Department of Computer Science
1215 Western Ave, UAB 424B Albany, NY 12222, USA
Email: cchelmis@albany.edu

***Abstract*—Online social networks offer a rich data source for analyzing diffusion processes including rumor and viral spreading in communities. While many models exist, a unified model which enables analytical computation of complex, non-linear phenomena while considering multiple factors was only recently proposed. We design an optimized implementation of the unified model of influence for vertex centric graph processing distributed platforms such as Apache Giraph.We validate and test the weak and strong scalability of our implementation on a Google Cloud Platform Hadoop and a Giraph installation using two real datasets. Results show a $\sim$ 3.2x performance improvement over the single node runtime on the same platform. We also analyze the cost of achieving this speedup on public clouds as well as the impact of the underlying platform and the requirement of having more distributed nodes to process the same dataset as compared to a shared memory system.**

1. Introduction

Online social networks offer a rich data source for analyzing diffusion processes such as rumor spread and information dissemination. Models to capture the rate and spread of such processes through complex and computationally expensive PDEs [1], or stochastic and probabilistic processes [2], [3] that unfold over the network based on some probability setting have been proposed. In the latter case, the models require a high number of simulation runs before statistically significant results can be achieved. Hence, there is a pressing need for efficient and scalable solutions to cope with the computational complexity of such models. Furthermore, computations involving sparse real-world graphs such as influence and information propagation in social media and viral marketing only manage to achieve a tiny fraction of the computational peak performance on the majority of current computing systems. The primary reason is that sparse graph analysis tends to be highly memory-intensive (i.e., data exhibits low degree of spatial and temporal locality, leading to a large memory footprint compared to other workloads, and there is very little computation to hide the latency to memory accesses. Thus, the execution time of a diffusion models strongly correlates with the memory performance, rather than the processor clock frequency or the number of processing units of the system. The design of efficient parallel graph algorithms is quite challenging as well, due to the fact that massive graphs that occur in real-world applications are often not amenable to a balanced partitioning among processors of a parallel system.

The unified model for rumor spread that was recently introduced in [4], enables the analytical computation of complex, non-linear phenomena such as influence, while considering multiple factors that can change over time without requiring extensive simulations to estimate the propagation probabilities at the steady state. However, as network datasets grow exponentially, cloud platforms such as those based on Apache Hadoop and MapReduce are increasingly being used to store and process such data. The reason to distribute the graph data and process it on the cloud lies not only in the memory limitations of existing systems (i.e., Twitter processes about 100 TB daily and Facebook around 600 TB daily[1]) but also in the distributed nature of the data itself with data centers holding information about social networks being geographically spread around the world. Such platforms are designed for distributed algorithms, and enable users to bring their application to the data, a task which becomes essential in the face of the data deluge social networks and other Big Data sources create. The challenge

1. https://followthedata.wordpress.com/2014/06/24/data-size-estimates/

978-1-5386-3472-1/17/$31.00 ©2017 IEEE

is therefore twofold: (1) to efficiency scale the computation of complex diffusion phenomena to massive datasets, and 2) to distribute the computation geographically.

The **objective** of this paper is to start from the shared memory algorithm proposed in [5] and to implement a distributed algorithm for computing the spread of a rumor in a small-world network based on the unified model [4]. Our distributed algorithm enables users to efficiently process data in clouds without being limited by existing single cluster systems. The **key idea** in the design of the algorithm is to leverage the individual graph node computation from the initially proposed shared algorithm and to design a vertex centric algorithm which can be easily distributed based on existing solutions such as Apache Giraph. We discuss this new algorithm in detail in Section 4.

Our key **contributions** are summarized below:

- We propose a distributed algorithm for the efficient computation of the unified model of social influence. We implement this algorithm using Apache Giraph;
- We examine the challenges posed by existing MapReduce solutions such as Giraph on a Google Cloud Platform when performing memory-intensive tasks;
- We compare the weak and strong scalability of our distributed algorithm against the previous shared memory implementation [5] on a cloud platform running on Google Cloud with 8 nodes each equipped with 2 vCPU.

The rest of the paper is structured as follows: Sect. 2 describes the most relevant recent work on parallel and distributed algorithms for social diffusion processes; Sect. 3 introduces the unified model of social influence; Sect. 4 presents the proposed distributed memory algorithm; Sect. 5 outlines our experiments and main results; and Sect. 6 summarizes our contribution and future work.

2. Related Work

While a lot of work has been done to model influence spread over networks (see the related work in [4] for more details) little work has addressed the issue of scaling up the computations on parallel and distributed systems as not only graphs constantly grow in size but the problems themselves get increasingly complex. In what follows we briefly give an overview of some representative recent work addressing shared and distributed memory algorithms for influence analysis.

Since the problem we are trying to solve (i.e., scaling the unified model of social influence) is relatively new, only a shared memory algorithm [5] exists. Specifically, an OpenMP parallel algorithm was previously proposed [5] that achieves ≈4–5.5 speedup compared to the sequential version on a large social graph containing around 3.8 million nodes and 5.4 million edges. In this paper, we modify the shared memory algorithm presented in [5], and propose a vertex-centric implementation that can run on distributed graph processing platforms on top of clouds.

An optimistic parallel execution of discrete event-based reaction-diffusion simulation models of epidemic propagation was discussed in [6]. Parallel agorithms were implemented using MPI and tested on both Bluegene/P and Cray XT5 machines, showing significant speedup, which however is dependent on both the specific hardware was and the tested scenario. In [7], an MPI algorithm for fast epidemic simulations was introduced. Results show that a 80-day simulation for a population of about 16 million, on a cluster using 96 Power5 processors, takes less than 5 minutes. Overall, the solution achieves a scalability of ∼3–4x for 224 processors relative to a baseline of 32 processors.

The emergence of graph processing tools such as Giraph [8] and other related platforms (e.g., Giraph++ [9], Apache Pregel [10], GoFFish [11]) drives the processing of large scale graphs on clouds. Already, the efficiency of some graph algorithms such as PageRank, connected components, single source shortest path, and random walks have been analyzed on various datasets [9], [11]. The main challenge with developing cloud based solutions is the overhead and configuration of underlying systems such as Hadoop. For instance, depending on the problem size, fine tuning of the heap and split size (logical data chunks used by MapReduce to start mappers) is required and the solution is usually custom tailored to the problem.

3. The Unified Model of Social Influence

The unified model of social influence was recently introduced in [4]. It is a generalized, analytical solution to the diffusion mechanism comprising of two processes unfolding over the network simultaneously: (a) pairwise influence, and (b) pressure from collective dynamics. The solution models each user independently and does not require extensive simulations, enabling fast simulation of personalized (time dependent) influence functions. Initial validation has considered several models such as the Independent Cascade Model (ICM) [12], the Linear Friendship Model (LFM) [13], and the Complex Contagion Model (CCM) [14].

In the unified model the social network is represented by a static directed graph where nodes are individuals and edges represent interactions. The model is expressed by a single formula which determines the probability of a node *u* to be infected at time *t* by its neighbors:

$$B_{u,t} = 1 - \Bigg[(1 - B_{u,t-1}) \left(\prod_{v \in N_i(u)} (1 - p_{v,u}(t) B_{v,t-1}) \right) + \left(\prod_{v \in N_i(u)} p_{v,u}(t) \, (1 - B_{v,t-1}) \right) \left(\prod_{k=1}^{t-1} (1 - r_u(k)) - 1 + B_{u,t-1} \right) \Bigg] (1 - r_u(t)) \quad (1)$$

where $N_I(u)$ is the set of incoming edges to node *u*; $p_{v,u}(t)$ represents the individual influence (i.e., the probability of

978-1-5386-3472-1/17/$31.00 ©2017 IEEE

node v infecting node u at time t), $r_u(t)$ is the collective influence independent of the individual influence. Initially, $B_{u,0}$ is either 0 or 1 depending on whether the node is infected or not.

The formulae for $p_{v,u}(t)$ and $r_u(t)$ are model specific. We give next some examples for ICM, LFM, and CCM:

- ICM [12]:

$$\begin{cases} p_{v,u}(t) = 0, \\ r_u(t) = 1 - \prod_{v \in N_i(u)} \left(1 - p\left(B_{v,t-1} - B_{v,t-2}\right)\right) \end{cases} \tag{2}$$

- LFM [13]:

$$\begin{cases} p_{v,u}(t) = 0, \\ r_u(t) = \frac{\exp(\alpha \sum_{v \in N_i(u)} B_{v,t-1} + \beta)}{1 + \exp(\alpha \sum_{v \in N_i(u)} B_{v,t-1} + \beta)} \end{cases} \tag{3}$$

- CCM [14]:

$$\begin{cases} p_{v,u}(t) = p, \\ r_u(t) = \exp(\alpha \sum_u B_{u,t-1} - \beta) \end{cases} \tag{4}$$

where p, α, β are constants.

The model assumes that infected nodes remain infected throughout the diffusion. The process repeats until a predefined stopping criterion is satisfied (e.g., number of time step, or fraction of infected nodes has exceeded a given threshold).

4. Distributed Algorithm for the Unified Model

The unified model of influence (Sect. 3) enables the probability of infection $B_{u,t}$ to be independently computed for each node u using Eq. 1. This maps nicely to the vertex-centric programming abstraction followed by graph processing platforms such as Apache Giraph and Google Pregel. These marry the ease of specifying uniform application logic for each vertex with the Bulk Synchronous Parallel (BSP) model to scale distributed graph applications. In the vertex centric abstraction each vertex processing takes place independently in a distributed environment with interleaved synchronized message exchanges. Each <processing, message exchange> sequence forms a superstep.

Figure 1 exemplifies the vertex centric abstraction mapped on the BSP model on a simplistic and straightforward algorithm. At every superstep each vertex sends its infection probability (either 0 or 1) to its neighbors. Upon receiving uninfected neighbors become infected, while already infected ones stay infected irrespective of the probability of infection by its neighbors.

Next, we detail our proposed algorithm for vertex centric influence propagation as outlined by the pseudocode in Fig. 2. Each compute method runs independently on every vertex and message exchanges happen after it completes. At each superstep the compute method can access the values stored in the vertex and messages sent in the previous step.

The challenge in our approach is to select which data to store locally and which to send to the next superstep. In the shared memory approach proposed in [5] the infection

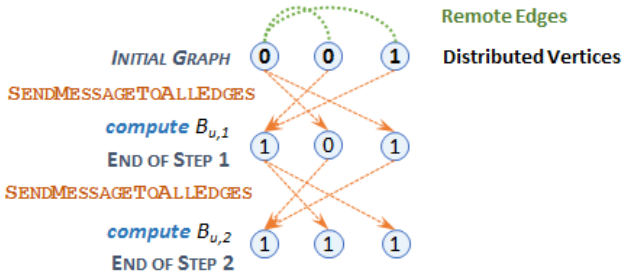


Figure 1. Example of node infection using the vertex centric approach.

```
1:  procedure COMPUTE(vertex, messages)
2:      t = getSuperStep()
3:      if vertex.isInfected ≠ 1 then
4:          B_{u,t-1} ← vertex.isInfected
5:          r_u(t) ← vertex.content ⊕ messages.content
6:          ∀v ∈ messages : p_{v,u} ← messages[v].content
7:          Compute B_{u,t} based on Eq. 1
8:      end if
9:      if t < MAX_SUPERSTEPS then
10:         message ← B_{u,t} and/or B_{u,t-1} depending on
        model
11:         sendMessageToAllEdges(vertex, message);
12:     else
13:         voteToHalt();
14:     end if
15: end procedure
```

Figure 2. Pseudocode of the proposed vertex centric influence propagation algorithm.

probability matrix B was stored in memory with each thread having access to it. This matrix is vital as $B_{u,t}$ depends on previous values either from the neighbors or from the vertex itself (cf. Eq. 1).

The naive solution would be to store the entire vertex history locally and to send to the neighbors at each step its probability of infection $B_{u,t}$. However, storing the entire history can easily outgrow the available memory on commodity cluster nodes as the number of supersteps and the graph size increases.

MapReduce chooses to parallelize tasks and start mappers based on the number of *splits*. A split is a logical user defined (by default it is 64MB) partitioning of the data where each split is processed locally by a map task. However, the placement of the data is dictated by HDFS and it is possible to run several mappers on the same physical machine.

In graph processing files usually contain the adjacency list and a default split can store several million nodes and edges. For example, given a default split it can hold ≈3 million vertices. Having the entire history of each vertex in memory can easily consume it. Considering float values around 32 million values can be stored in one GB. The history created during each superstep on a 8 GB memory machine is filled after only 85 supersteps assuming nothing else takes up memory which is not the actual case.

978-1-5386-3472-1/17/$31.00 ©2017 IEEE

To overcome these limitations we can to store in each vertex and message only the values needed by the infection model to compute the next infection probability. Specifically, for each of the three discussed models we can store the following information:

- ICM:
  - vertex content: the last two values for B, ($B_{u,t-1}$ and $B_{u,t-2}$);
  - message content: the vertex content and $p_{v,u}$. This will be used in computing Eq. 2 before inserting it into Eq. 1.
- LFM:
  - vertex content: the value $B_{u,t-1}$;
  - message content: the vertex content and $p_{v,u}$. This will be used in computing Eq. 3 before inserting it into Eq. 1.
- CCM:
  - vertex content: the value $B_{u,t-1}$;
  - message content: the vertex content and $p_{v,u}$. This will be used in computing Eq. 4 before inserting it into Eq. 1.

In addition, in order to avoid keeping the entire node history for each vertex we only store the the impact of its collective influence $\prod_{k=1}^{t-1}(1 - r_u(k))$ cf. Eq. 1.

The algorithm (cf. Fig. 2) proceeds only if the node is not infected (i.e., $B_{u,t-1} \neq 1$ – cf. Line 3) otherwise it only sends its infection probability to other nodes (cf. Line 11).

Then, the collective influence $r_u(t)$ (cf. Line 5) is computed by applying the model specific formula (e.g., one of the equations 2, 3, 4 or from other models). Relevant information is extracted from the message and vertex content (see previous description of what they store).

The individual influence $p_{v,u}$ (cf. Line 6) is also computed based on information stored in the message received from neighboring nodes v (for the three models we consider this value is either 0 or a constant but others can have more complicated formulae).

Finally, its infection probability at current time *t* is computed in Line 7 and transmitted to the neighbors (cf. Line 11).

5. Performance Analysis

We present a detailed analysis of the performance of our distributed algorithm using the shared memory implementation that was discussed in [5] as our reference. For comparison, in [5] an 8 core 2.66 GHz machine was used for the OpenMP shared memory experiments.

Setup. We report performance results on the following cloud configuration consisting of 9x Google Compute Cloud n1-standard-2 instance with 2 vCPU (Intel Haswell) and 7.5 GB RAM. One machine is used as the namenode while the other 8 are for processing. Nodes run Ubuntu 16.04 OS with Hadoop 2.7.1, Giraph 1.1.0-hadoop2, and Zookeeper 3.4.10. Google Cloud the n1 series of machines implement a virtual CPU as a single hardware hyper-thread on Intel Xeon E5 machines ranging between 2.2-2.6 GHz[2]. The actual choice is not visible for the end user. At the time of the experiments we did have access to larger cloud infrastructures.

Each experiment was run three times and the average values were considered for plotting and analysis. We noticed that due garbage collection limits (which we could not modify for the current setup) we are unable to run larger graphs on few number of nodes. To partially solve this issue we used single precision instead of double for computing the values of **B** which had enabled us for instance to run the distributed algorithm on a 250,000 nodes graph on 4 machines. When using double precision this was not possible. For reproducibility of our results, the source code of our algorithm is available on GitHub[3]. Since the only difference between models lies in the formulas for $p_{v,u}$ and $r_u(t)$ and not the number of messages and the algorithm itself we opted to test in our experiments only the CCM model.

Datasets. For fairness of comparison, we have evaluated our algorithm on the same real-world datasets used in [5]:

- HEPT Collaboration network (same as used by [5], [15]) from the "High Energy Physics - Theory" section of arXiv4 with papers form 1991 to 2003 and is denoted as HEPT. It contains 15,233 nodes and 58,891 edges;
- Twitter network[4]. We considered there is a directed link between two users if one user has mentioned the other in a tweet. This network has around 2.3 million nodes and around 5.4 million edges. To test both weak and strong scalability we have sampled the Twitter graph randomly at various sizes ranging from 50K to 700K nodes.

Week and Strong Scaling. For our analysis, we use as reference the speedups reported in [5] for HEPT (3–4x) and Twitter (4–5.5x) on an 8 core shared memory machine. All three models achieved a similar speed-up for the strong scaling case. No weak scaling experiments were performed in the cited paper. We summarize our weak and strong scalability tests in Fig. 3 for the Twitter datasets. In case of HEPT the graph was too small and the platform overhead too high to achieve any scalability. Figure 4 show the runtimes for both Twitter dataset with various number of graph nodes processed and the full HEPT dataset. Figure 5 shows the percentage of time spent by each step in running the algorithm for both Twitter and HEPT. We analyze the results in detail next.

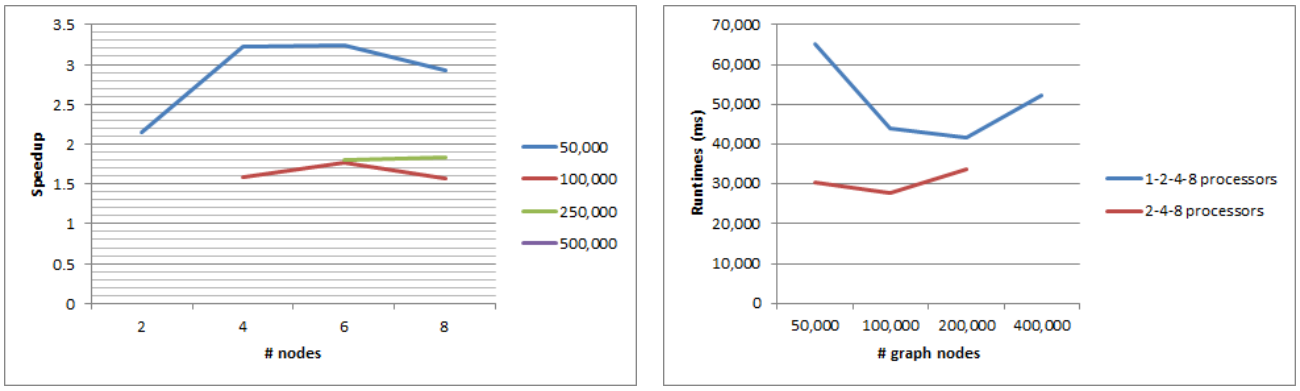
Runtime. The 700,000 node Twitter graph runs for ∼230,078ms. Due to garbage collection memory limits we were unable to run it on less than 8 nodes. The same was true for smaller graphs on fewer nodes as shown in Fig.

2. https://cloud.google.com/compute/docs/machine-types (accessed on May 23, 2017)

3. Source code available at https://github.com/alecspopa/rumorspread-giraph

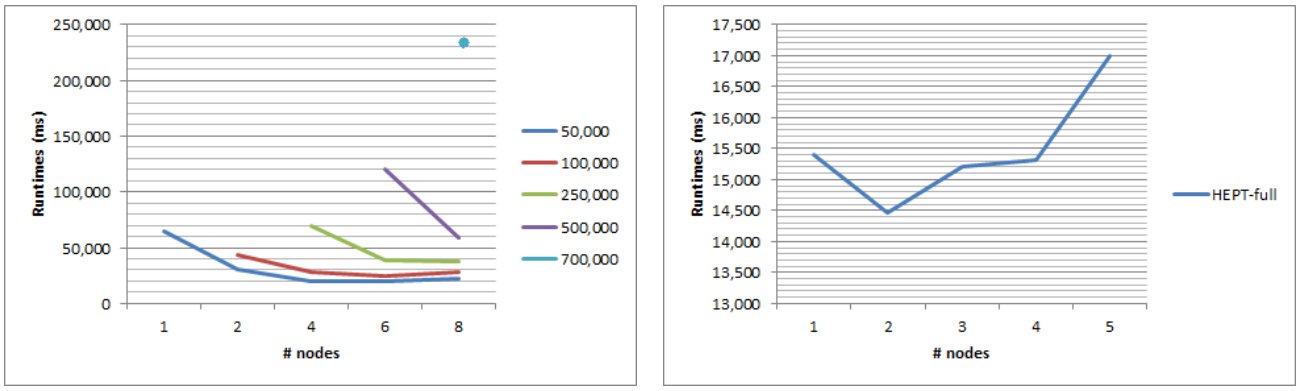
4. Available at http://trec.nist.gov/data/tweets/

978-1-5386-3472-1/17/$31.00 ©2017 IEEE



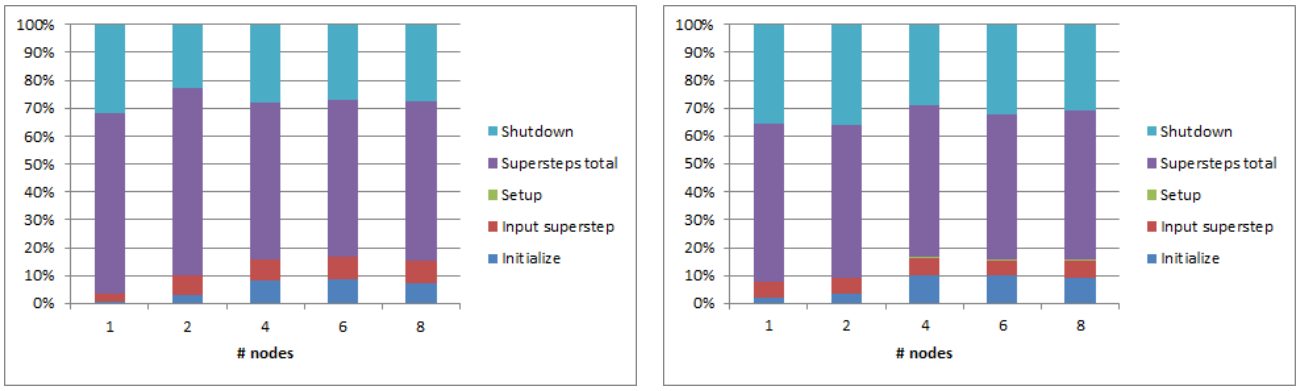
(a) Strong scaling (b) Weak scaling

Figure 3. Scaling results for the Twitter dataset.



(a) Twitter (b) HEPT

Figure 4. Runtime results in ms for both datasets.



(a) Twitter (b) HEPT

Figure 5. Time partitioning results for both datasets.

4. Limited access to the cloud infrastructure at the time of the experiments did not allow us to test on more nodes the runtime of the entire graph.

In the case of HEPT (which is a graph of only 15K nodes), increasing the number of nodes leads to an increase in the runtime due to communication overhead. Our runtimes do not consider platform overhead which we individually measured in Fig. 5. As showed in Fig. 5, platform overhead accounts for ∼45% of the total time for both the smaller HEPT and the larger Twitter datasets.

Scalability. Scalability tests used as baseline the first configuration that worked. As an example for the 100K node

978-1-5386-3472-1/17/$31.00 ©2017 IEEE

Twitter graph we referred to the runtime of the algorithm on the setup with 2 nodes as fewer machines could not execute it.

We notice a peak strong scalability of ≈3.2x and that the communication overhead quickly takes its toll when increasing the number of machines (cf. Fig. 3a). In the future we intend on further fine tuning the Hadoop and Giraph setup to increase the garbage collection limits enabling us to run larger graphs on fewer nodes as well. Overall we notice a slower scalability (≈3x compared to ≈4–5.5x) than in the case of the shared memory algorithm which is to be expected given the communication overhead of running in a distributed environment. However, despite a lower speedup the proposed algorithm shows the potential of distributed graph analytics for social influence analytics on clouds.

For the weak scaling tests we wanted to see if the runtime stays stable as the graph size and number of machines doubles. As it can be seen from Fig. 3b the runtimes are relatively stable especially when using the 2 machine setup as baseline when doubling graph and number of machines (2 machines for the 50K graph; 4 machines for the 100K graph; and 8 machines for the 200K graph). It should be noted that in this case we do not have results for 16 machines processing the graph of 400K nodes. When the baseline is the execution on one machine (1 machine for the 50K graph; 2 machines for the 100K graph; 4 machines for the 200K graph; and 8 machines for the 400K graph) we notice a standard deviation of ≈9,000 ms from the average value of ≈50,000 ms.

Cost. When running on public clouds, the issue of cost inevitably comes up. In our case, the algorithm achieves ≈3x speedup at 6 times the cost of the single node configuration in the case of the Twitter dataset (which is the larger of the two datasets that we experimented with). For HEPT, the platform overhead means that it is better to run it on a single node as no noticeable speed-up was observed. Intuitively, our proposed distributed algorithm is well suited for large datasets that do not naturally fit in memory of a single node.

6. Conclusion

In this work we have introduced a vertex centric distributed algorithm for he unified model of social influence. We have demonstrated a ≈3.2x speedup compared to a single node Giraph installation and weak scaling of our solution and have analyzed the cost impact on public clouds. At the same time we noticed the impact of the network overhead and underlying platform which requires fine tuning of parameters to enable larger graphs to be executed on fewer nodes. Similarly, simply by scaling down the accuracy of the computations from double to single precision has the same effect. This is possible as single precision suffices when computing the probability of infection.

We emphasize here two main conclusions from our experiments which we intend on pursuing as future work. First, our current setup requires far more nodes than the shared memory to run the same graph. For instance, with 8 machines we were able to run only 26% of the entire dataset compared to the shared memory algorithm which executed it entirely. We will focus our efforts in fine tuning the platform to increase scalability. Second, the communication overhead becomes a bottleneck fast capping the speedup and the distributed nodes we can use. We believe this to be mainly caused by the structure of the analyzed graph as each computation is done per vertex and the only communication happens between neighbors. We intend to perform tests on other BSP based models such as the subgraph centric approach proposed in [11] which enable us to partition the graph and process local partitions without the inducing network overhead.

Acknowledgments

The authors would like to thank Bogdan Butunoi from West University of Timisoara for managing the installation and setup of the Hadoop cluster and Apache Giraph, and Ajitesh Srivastava from University of Southern California for providing the codes for the OpenMP version as well as clarifications about the algorithm.

References

[1] K. Rock, S. Brand, J. Moir, and M. J. Keeling, "Dynamics of infectious diseases," *Reports on Progress in Physics*, vol. 77, no. 2, p. 026602, 2014.

[2] A. Hajibagheri, A. Hamzeh, and G. Sukthankar, "Modeling information diffusion and community membership using stochastic optimization," in *2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2013)*, 2013, pp. 175–182.

[3] J. Jacquez and C. Simon, "The stochastic si model with recruitment and deaths i. comparison with the closed sis model," *Mathematical Biosciences*, vol. 117, no. 1-2, pp. 77–125, 1993.

[4] A. Srivastava, C. Chelmis, and V. K. Prasanna, "The unified model of social influence and its application in influence maximization," *Social Network Analysis and Mining*, vol. 5, no. 1, p. 66, 2015.

[5] ——, "Computational models for cascades in massive graphs: How to spread a rumor in parallel," in *Parallel Graph Algorithms*. Chapman & Hall/CRC, 2016, p. in print.

[6] K. S. Perumalla and S. K. Seal, "Discrete event modeling and massively parallel execution of epidemic outbreak phenomena," *Simulation*, vol. 88, no. 7, pp. 768–783, Jul. 2012.

[7] K. R. Bisset, J. Chen, X. Feng, V. A. Kumar, and M. V. Marathe, "Epifast: A fast algorithm for large scale realistic epidemic simulations on distributed memory systems," in *Proceedings of the 23rd International Conference on Supercomputing*, ser. ICS '09, 2009, pp. 430–439.

[8] A. Ching, S. Edunov, M. Kabiljo, D. Logothetis, and S. Muthukrishnan, "One trillion edges: Graph processing at facebook-scale," *Proc. VLDB Endow.*, vol. 8, no. 12, pp. 1804–1815, Aug. 2015.

[9] Y. Tian, A. Balmin, S. A. Corsten, S. Tatikonda, and J. McPherson, "From "think like a vertex" to "think like a graph"," *Proc. VLDB Endow.*, vol. 7, no. 3, pp. 193–204, 2013.

[10] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: A system for large-scale graph processing," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '10, 2010, pp. 135–146.

[11] Y. Simmhan, A. Kumbhare, C. Wickramaarachchi, S. Nagarkar, S. Ravi, C. Raghavendra, and V. Prasanna, *GoFFish: A Sub-graph Centric Framework for Large-Scale Graph Analytics*. Springer International Publishing, 2014, pp. 451–462.

[12] D. Kempe, J. Kleinberg, and E. Tardos, "Maximizing the spread of influence through a social network," in *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '03, 2003, pp. 137–146.

[13] A. Anagnostopoulos, R. Kumar, and M. Mahdian, "Influence and correlation in social networks," in *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '08, 2008, pp. 7–15.

[14] C. Chelmis and V. K. Prasanna, "The role of organization hierarchy in technology adoption at the workplace," in *2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2013)*, 2013, pp. 8–15.

[15] W. Chen, Y. Wang, and S. Yang, "Efficient influence maximization in social networks," in *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '09. ACM, 2009, pp. 199–208.

978-1-5386-3472-1/17/$31.00 ©2017 IEEE