Proceedings of the 2014 Winter Simulation Conference
A. Tolk, S. Y. Diallo, I. O. Ryzhov, L. Yilmaz, S. Buckley, and J. A. Miller, eds.

HANDLING BIG DATA ON AGENT-BASED MODELING OF ONLINE SOCIAL NETWORKS WITH MAPREDUCE

Maíra A. de C. Gatti
Marcos R. Vieira
João Paulo F. de Melo
Paulo Rodrigo Cavalin
Claudio Santos Pinhanez

IBM Research – Brazil
Avenida Tutóia 1157
São Paulo, 04007-005, BRAZIL

ABSTRACT

There is an increasing interest on using Online Social Networks (OSNs) in a wide range of applications. Two interesting problems that have received a lot of attention in OSNs is how to provide effective ways to understand and predict how users behave, and how to build accurate models for specific domains (e.g., marketing campaigns). In this context, stochastic multi-agent based simulation can be employed to reproduce the behavior observed in OSNs. Nevertheless, the first step to build an accurate behavior model is to create an agent-based system. Hence, a modeler needs not only to be effective, but also to scale up given the huge volume of streaming graph data. To tackle the above challenges, this paper proposes a MapReduce-based method to build a modeler to handle big data. We demonstrate in our experiments how efficient and effective our proposal is using the Obama's Twitter network on the 2012 U.S. presidential election.

1 INTRODUCTION

Nowadays Online Social Networks (OSNs) are one of the most used tools for information diffusion. Information diffusion is a model for spreading a new idea or action through communication channels (Rogers and Rogers 2003), which has been heavily studied by sociologists, marketers, and epidemiologists (Kempe, Kleinberg, and Tardos 2003; Leskovec, Adamic, and Huberman 2006; Strang and Soule 1998). In this context, large OSNs are useful, for example, for studying information diffusion as topic propagation in blogspace (Gruhl et al. 2004), linking patterns in blog graph (Leskovec and Horvitz), favorite photo marking in a social photo sharing service (Cha, Mislove, and Gummadi 2009), among many other domains.

Modeling users' behavior on large OSNs has become a very interesting problem due to the variety, complexity, and abundance of data that online media has. Agent-based simulation is one type of modeling that is consistent with the sciences of complexity (Jennings 2001). Agents are autonomous entities capable of acting without direct external intervention. Multi-agent systems can handle the complexity of solutions through decomposing, modeling and organizing the interrelationships between components. Hence, we can model each user behavior independently based on the available OSN data.

In previous work we have shown how to gather sentiment information about topics from the content generated by users. We also demonstrated how to model users' behavior based on their generated content (Gatti et al. 2014). Furthermore, we have analyzed how key nodes in the network influence the information propagation and what are the effects when these key nodes are dropped from the network (Gatti et al. 2013). We observed that users show a quick response to stimulae regarding the seed's topic in an

egocentric network. This finding shows that a significant share of the user's behavior can be captured considering short time windows, even when using a simple first order Markov Chain (Neto et al. 2013).

In this context, big data plays a very important role when modeling OSN. Twitter, one of the most used microblogging OSN, reported in 2013 more than 550 million active registered users and on average 58 million tweets posted per day. Dealing with this large amount of data is not an easy task. In this work we propose a solution based on the MapReduce framework (Dean and Ghemawat 2008) to build agent-based models from massive amounts of data and very large networks. For evaluating the proposed method, we have conducted experiments on a large sample of Barack Obama's Twitter network gathered during the 2012 U.S. presidential elections. Using simulations in several scenarios with different graph sizes, we demonstrated that the modeler can scale up well to large graph data.

2 PROBLEM DESCRIPTION

There exist three types of parallel architectures it is worthy mentioning: shared memory, shared disk, and shared nothing systems (DeWitt and Gray 1992). In shared memory systems every process has access to all the memory and disk. Shared disk is an architecture where the file system is shared across different processes with no shared memory. Shared nothing is when individual machines are only connected by networks, thus no shared memory or shared disk are available. The main advantage of shared nothing clustering is scalability, i.e., (in theory) it can scale up to thousands of nodes since they do not interfere with one another.

MapReduce (Dean and Ghemawat 2008) is designed for shared nothing architectures. It is a programming model for expressing distributed computations on massive amounts of data and a lightweight execution framework for large scale data processing on clusters of commodity servers. It provides automatic parallelization and distribution, fault-tolerance, I/O scheduling, and status and monitoring. In the MapReduce architecture, Distributed File Systems (DFS), like HDFS[1], is essential for very large files – terabytes or petabytes – where files are split into chunks and replicated through distributed disks.

In agent-based modeling of OSN, since there are many tasks processing and producing large volumes of data, it is often required a large-scale data processing architecture. In order to scale up the execution of these tasks, many agent-based modeling architectures use hundreds, or even thousands, of CPUs to achieve a high sampling size of the social network. In this scenario, there are two data processing approaches for multi-agent system when dealing with large volumes of data: the in memory processing approach and the data distributed processing approach. Each solution has its own drawbacks with respect to non-functional requirements, like fault-tolerance and adaptation (i.e., there is always a tradeoff to achieve fast execution and fault tolerance).

(Gatti et al. 2014) proposed an agent-based engineering method to model and simulate OSN that does not tackle large-scale data processing. This method is based on a stochastic multi-agent based approach where each agent is modeled as a Markov Chain process integrated with a Monte Carlo simulation and historical data of each user in the network. This method is an iterative process and is composed of six phases:

1. **Sampling network and nodes' behaviors**: sampling and cleaning tasks are performed on the OSN data;
2. **Topic and sentiment classification**: consists of performing topic and sentiment classification on the extracted posts from the sampled data;
3. **Dataset partitioning**: sets of samples are created for each user from the previously classified data. Each set contains the user's posts and the posts of whom he/she follows;

[1]HDFS stores large files and achieves reliability by replicating the data across multiple hosts. It does not require RAID storage on hosts, but to increase I/O performance some RAID configurations are still useful. With the default replication value (i.e., 3 replicas) data is stored on three nodes: two on the same rack and one on a different one. Data nodes can communicate to each other to rebalance data, to move copies around, and to maintain the replication of data.

4. **Users' (nodes) behaviors predictive modeling**: builds each user's behavior model from the sets of samples;
5. **Simulator building**: the previous models are validated using the stochastic simulator;
6. **Forecast information diffusion**: the previous steps are performed several times, until the model is accurate enough. Then the forecast on information diffusion can be performed.

The following issues should be considered when performing steps 3 and 4:

- **Memory size**: datasets are to large to fit in memory, and thus must be held in disk;
- **Real-time updates both in the graph and the users' activities**: how the models are updated while streaming data continuously comes in?
- **Fault-tolerance**: how to overcome the failure of one process?
- **Scalability**: what is the tradeoff between scaling up and scaling out with an in memory or distributed solution?

Although in this paper we are not tackling the real time challenge, we consider that model updates might happen during the modeler execution, since the modeler state and the agents' states that are modeled can be stored in the HDFS.

2.1 Predictive Modeling of Users' Behavior

In the following we describe how user's behavior can be modeled and predicted based on simple actions, like reading and writing as proposed in Gatti et al. (2014). There exist other approaches that could be considered as well, but it is out of the scope of this paper.

In this work we aim at learning only the most important user's action in an OSN, i.e., the *posting* action. To do so, this modeler receives as input the list of users and, for each user, a document containing the user's posts and the posts of whom he/she follows in the network. With this information, the user's behavior is modeled by means of state transitions of a First-Order Markov Chain. The current state depends only on the previous state, which is based on the message topics of what he/she posted and his/her connections.

To build the modeler, we consider the following three premises:

1. Time is discrete and time interval δt defines an action time window at time t;
2. User's actions (e.g., posting) are performed on δt with the correspondent states. The current state A_t of the modeler defines what the user posted in the current time window, while previous state $A_{t-\delta t}$ defines that he/she posted and/or read in the previous time window;
3. Messages are interpreted as a bit vector representing whether the topic appears or not in the message.

Let R be the vector representing what the user read (i.e., whom he/she follows posted), W what he/she wrote, the previous and current states are defined, respectively, as $A_{t-\delta t} = \{R_{t-\delta t}, W_{t-\delta t}\}$ and $A_t = \{W_t\}$. Table 1 describes the 7 different types of transitions and/or states that can be observed in the data used in the simulator. The empty sets $R_{t-\delta t} = \emptyset$, $W_{t-\delta t} = \emptyset$ and $W_t = \emptyset$ represent non-observed vectors.

To estimate the transition type parameter $\theta_i \in \theta$ the Maximum Likelihood Estimation (MLE) with smoothing is computed. Thus, for each user's sampled data u, $\mathscr{L}$ is estimated for:

- Observed transitions θ_1, θ_3, θ_5:

$$\mathscr{L}(\theta | R_{t-\delta t}, W_{t-\delta t}, W_t) = \frac{count(\theta, R_{t-\delta t}, W_{t-\delta t}, W_t) + 1}{count(R_{t-\delta t}, W_{t-\delta t}, W_t) + |S|} \tag{1}$$

- Non-observed transitions θ_2, θ_4, θ_6 and θ_7:

$$\mathscr{L}(\theta | R_{t-\delta t}, W_{t-\delta t}, W_t) = \frac{1}{count(R_{t-\delta t}, W_{t-\delta t}, W_t) + |S|} \tag{2}$$

Table 1: List of observed states and transitions.

θ	**Transitions**	
1	$R_{t-\delta t}$, $W_{t-\delta t} \neq \emptyset$	$\rightarrow W_t \neq \emptyset$
2	$R_{t-\delta t}$, $W_{t-\delta t} \neq \emptyset$	$\rightarrow W_t = \emptyset$
3	$R_{t-\delta t} = \emptyset$, $W_{t-\delta t} \neq \emptyset$	$\rightarrow W_t \neq \emptyset$
4	$R_{t-\delta t} = \emptyset$, $W_{t-\delta t} \neq \emptyset$	$\rightarrow W_t = \emptyset$
5	$R_{t-\delta t} \neq \emptyset$, $W_{t-\delta t} = \emptyset$	$\rightarrow W_t \neq \emptyset$
6	$R_{t-\delta t} \neq \emptyset$, $W_{t-\delta t} = \emptyset$	$\rightarrow W_t = \emptyset$
7	$R_{t-\delta t} = \emptyset$, $W_{t-\delta t} = \emptyset$	$\rightarrow W_t \neq \emptyset$

where $|S|$ is the number of states. The aforementioned transitions are computed for each topic $\omega_i \in \omega$ so that the users' actions are modeled according to the type of message he/she reads/writes.

In addition, the user might behave differently according to the period of the day. The probability of posting a message at a given period $\phi_i \in \phi$ is also computed, where k, $1 \leq i \leq k$, is the number of periods pre-defined prior to the modeling. This takes into account the total messages m_i posted by the user at ϕ_i, and the messages posted over all periods (the entire day), as in Equation 3. The following notation is used for each period ϕ_i. The corresponding starting time is denoted by $\phi'_i \in \phi'$, and its length (in hours) is denoted by $|\phi_i|$.

$$\mathscr{L}(\text{posting}|\phi_i) = \frac{m_i}{\sum_{\phi_j \in \phi} m_j} \tag{3}$$

2.2 The Case Study

As a case study, we crawled real data from the Twitter OSN. In the first phase of the engineering method previosly described, we developed a variation of the snowball algorithm to sample an egocentric network (Gatti et al. 2013): a network that considers a node as focal point and its adjacency. Our central node for this work is Barack Obama's Twitter account.

The crawling process consisted of two main phases. In the first phase, which took place during the week starting on October 26, 2012, we extracted the information about the structure of the network, resulting in a network with approximately 32 million nodes. During the second phase we collected the posts, from September 22, 2012 to October 26, 2012, where we ended up with data for time window of about one month. Table 2 shows more details on the extracted data.

Table 2: Sampled data and graph.

Description	**Size**
Tweets	5.6M
Active users	24,526
Obama's direct followers	3,594 (0.017% of real amount)
Edges	160,738
Triangles	83,751

Before we started the modeling, all the messages were classified into a pre-defined set of topics. First we selected a list of keywords to represent each topic. Next, each tweet message was tokenized using blank spaces and punctuation marks as separators. Then, the tokenized tweets were classified as belonging to one of the following topics:

- The tweet is classified as belonging to a particular topic if it contains at least one keyword from the topic list;

- The tweet is classified as *other* topic if it does not contain any keyword from any topic;
- The tweet is discarded if it contains keywords from more than one topic;

We considered only one topic, the Obama's 2012 presidential campaign, and the keyword list includes the words: *barack, barack2012, barackobama, biden, joebiden, josephbiden, mrpresident, obama, obama2012, potus*.

2.3 In Memory-Based Solution

We implemented an in memory-based solution for the modeling approach described in the last section. This implementation receives two files as input: one with the graph structure $G = (V, E)$ and the second one containing the user's posts and the posts of all his/her followers in chronological order. The modeler iterates over each post and populates the hashtables to compute the read/write vectors. Considering $|V|$ the number of users in the graph, $|E|$ the number of edges, and p the size of the input containing the users' posts and the posts of all their follows, the cost of our implemented algorithm for estimating the parameters in Equations 1 and 2 is $O(|E| * p)$. This algorithm has two main drawbacks: **(1)** it may be infeasible for the modeler to process users that have many posts and many followers; and **(2)** the modeler has to run again whenever new activities are observed.

Table 3 shows the running time and memory usage when running some experiments on a node with 16 CPU cores and 256GB of memory with a Red Hat x86_64 Linux. In order to run the tests, we processed the input files containing the users' activities by decreasing value of the absolute difference between file size and mean file sizes. This approach guarantees that the running time was not affected by the users' degree of activity.

Table 3: Memory usage and running time of the in memory-based solution when increasing network size.

Network size	Memory usage (GB)	Running time (min)
100	8.31	1
1,000	9.59	9
10,000	10.92	83
24,526	17.69	237.5

Even though the running time grows as a linear factor of the network size, memory requirements increase exponentially. Note that, in order to increase the accuracy of the modeler we need to have the input network size as big as its real size. In Obama's network, for instance, the number of followers at the time of the experiments was about 23 million users. To have a more complete level, the network size including the second level (i.e., 23 million users, their followers, and their followers' followers) would reach up to billions of users. In this scenario, a single server would not be able to process such amount of data owing to memory limitations. After fitting a polynomial curve of order 2 on the memory data, we estimated that the aforementioned server would be able to process a network size of approximately up to 152K users.

3 OUR PROPOSED MAPREDUCE-BASED MODELING APPROACH

In this section we describe our proposed MapReduce-based approach to handle very large OSN. Figure 1 shows a general overview of the MapReduce- based solution. The system receives the user's graph, which is an egocentric network with distance 1, and the user's activities in the OSN. The activities can be composed of the post information or other activities as if the user liked, forwarded, replied a post or any perceived interaction. The system fuses these inputs for all users in the sample and forwards it to the *User Behavior Modeler*. The results can be extracted from the HDFS and persisted in a database to be further analyzed

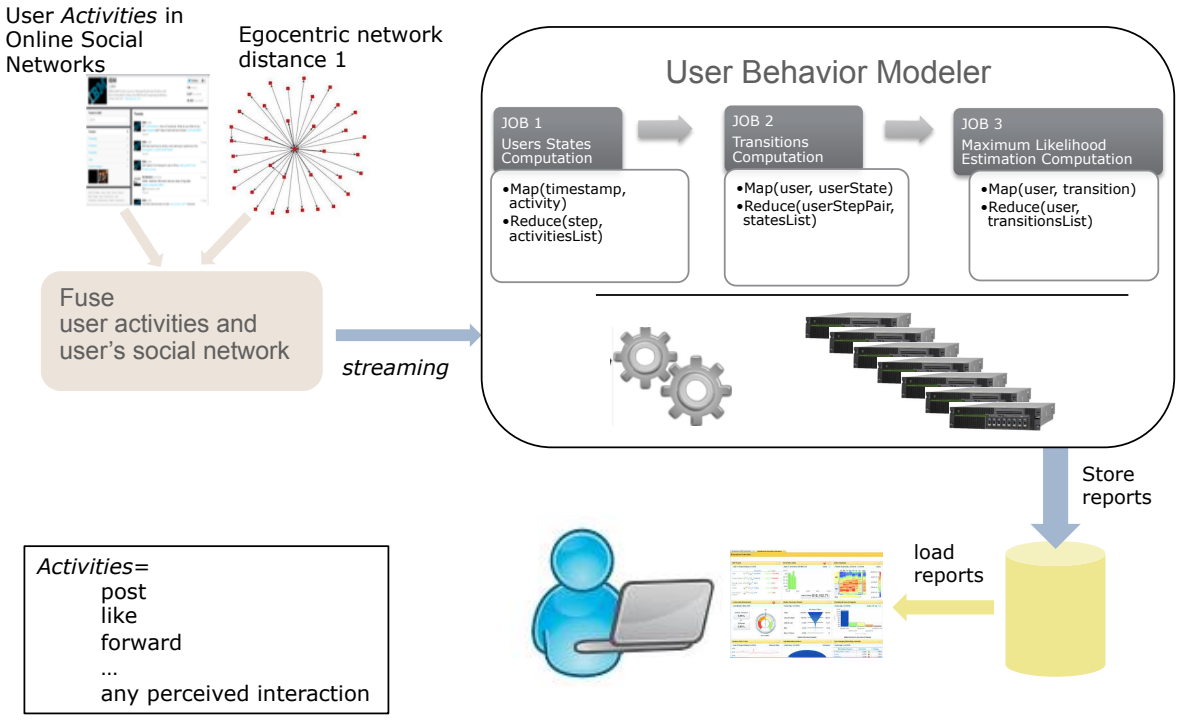


Figure 1: Overview of the MapReduce-based modeling approach.

by data scientists or users. In this case, the data is used as an input for the multi-agent-based simulator on which what-if analysis can be performed.

The *User Behavior Modeler* encapsulates the MapReduce solution. The MapReduce workflow divides the computation in separate jobs, reducing the complexity of developing and maintaining the modeler implementation. Also, it allows to work with samples without having the need to worry about memory usage and performance for larger data sets. Once the solution is running in a Hadoop cluster framework, it is easy to run the same solution on a much larger cluster environment. Our proposed approach consists of three connected jobs, as illustrated in Figure 2.

3.1 Job 1: Users' States Computation

The first job, described in Algorithm 1, is designed to compute user's states, as described in Section 2.1. The files containing the network graph and the users to be modeled are stored in *DistributedCache* for later use. The main input is the file containing the classified activities, which are mapped to time intervals defining action time windows.

In the reduce phase vectors containing user states are computed based on what was written/read for each topic. This is the most crucial part of the MapReduce-based modeler since we need to know whether the user actions are of a user we are modeling or are of a user that is followed by our users of interest. This verification phase substantially increases the time complexity of our algorithm. The output of the first job is a pair $<key,value>$, where key is the user id and value is the user state vector.

Table 4 shows an example of the input data for Job 1. An action time *step* can represent one or more lines and the *step* are mapped to a list of tuples with activities as:

$step_1 : [\{u_1,t_1,s_1\},\{u_1,t_2,s_1\},\{u_1,t_1,s_2\},\{u_2,t_1,s_1\},\{u_2,t_2,s_2\},...]$

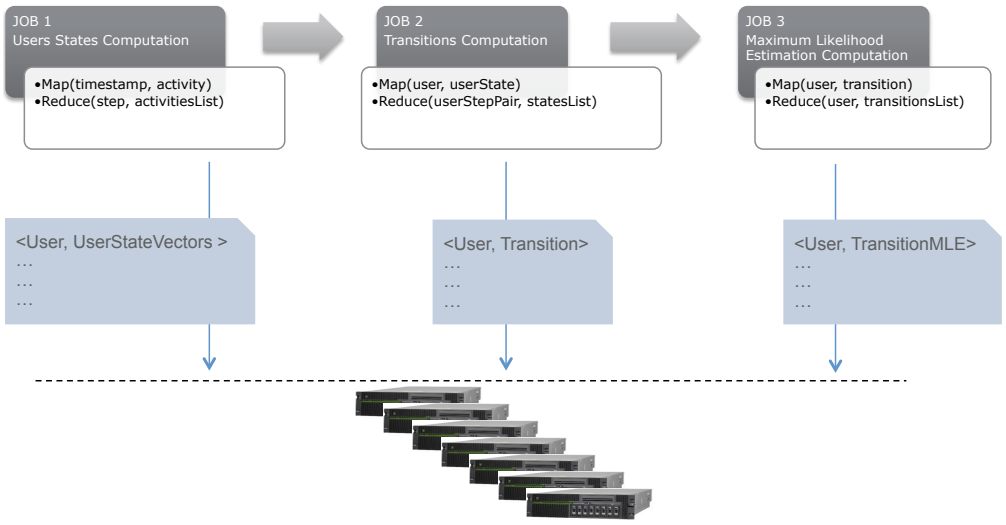


Figure 2: The workflow of the three jobs. Job 1: users' states computation; Job 2: transitions computation; Job 3: maximum likelihood estimation computation.

Algorithm 1 Users states job algorithm.

```
1: function MAP(timestamp, activity)
2:     step ← (parseDate(timestamp) - firstObservedTimestamp) / δt;
3:     emitIntermediate(step, activity);
4: end function
5: function REDUCE(step, activitiesList)
6:     for each activity in activitiesList do
7:         if activity.user in usersOfInterest then
8:             userPartialState ← new State(activity.user);
9:             userPartialState.analyzeAction(Action.WRITTEN, activity.topic, activity.sentiment);
10:            computeState(userPartialState);
11:        end if
12:        followers ← graph.getFollowers(activity.user);
13:        for each follower in followers do
14:            if follower in usersOfInterest then
15:                userPartialState ← new State(activity.user);
16:                userPartialState.analyzeAction(Action.READ, activity.topic, activity.sentiment);
17:                computeState(userPartialState);
18:            end if
19:        end for
20:    end for
21:    for each state in stepStatesMap do
22:        emit(state.user, toString(state.stateVectors));
23:    end for
24:    stepStates.clear();
25: end function
```

3.2 Job 2: Transitions Computation

The second job, described in Algorithm 2, calculates the transitions between user states and how many times the transition occurred. This jobs receives as input the generated <*user,userState*> from Job 1, and then maps it using the user as key. The reducer job receives a set of states for each user. We ensure that

Table 4: Input example for mapper in Job 1, where *activity* is the tuple [*user, topic, sentiment*].

Date time	User	Topic	Sentiment
dt_1	u_1	t_1	s_1
dt_2	u_1	t_1	s_2
...	...	...	...
dt_n	u_5	t_2	s_1

Algorithm 2 Transitions job algorithm.

```
1: function MAP(user, userState)
2:     userStepPair ← new UserStepPair(user, userState.step);
3:     emitIntermediate(userStepPair, userState);
4: end function
5: function REDUCE(userStepPair, statesList)
6:     for each state in statesList do
7:         if previousState = null then
8:             computeTransitionsUntilCurrentState(userState);
9:             previousState ← userState;
10:        else
11:            if previousState.step + 1 = userState.step then
12:                transition ← new Transition(previousState, userState);
13:                computeTransition(transition);
14:                previousState = userState;
15:            else
16:                computeTransitionsUntilCurrentState(userState);
17:                transition ← new Transition(previousState, userState);
18:                computeTransition(transition);
19:                previousState ← userState;
20:            end if
21:        end if
22:    end for
23:    for each transition in userTransitionsMap do
24:        emit(userStepPair.user, toString(transition));
25:    end for
26:    userTransitionsMap.clear();
27: end function
```

the set of states is sorted by using a composite key and a custom partitioner, implementing a technique commonly known as *secondary sort*.

The mapper receives pairs of <*user,userState*>, and for each pair it combines *user* and *step* and returns a pair <*userStepPair, userState*>. Then we shuffle by the user key, and sort by the pair <*user,step*>. The reducer finally counts the occurrences of each transition per user.

We can compute transitions from a state to another one if its time interval id is sequential. If this condition is not met, it means that the user has gone into an idle state and, therefore, idle transitions until the next observed state. The output consists of a pair containing the user as the key and a transition object containing its frequency as the value.

3.3 Job 3: MLE Computation

The third job, as described in Algorithm 3.3, computes the MLE with smoothing to estimate the parameter for a given transition. This jobs receives as input the generated <*user,transition*>. The reduce phase gathers the variables for the computation and generates for each user the pairs containing the transitions and the MLE.

Algorithm 3 Maximum Likelihood Estimation job algorithm.

```
1: function MAP(user, transition)
2:     emitIntermediate(user, transition);
3: end function
4: function REDUCE(user, transitionsList)
5:     for each transition in transitionsList do
6:         computeMLE(transition);
7:     end for
8:     for each transitionMLE in transitionsMLEMap do
9:         emit(user, toString(transitionMLE));
10:    end for
11:    transitionsMLEMap.clear();
12: end function
```

With the output of the third job, the user behavior models can be used as input for the agents in the simulator. Thus, the fifth *step* of the engineering method can be performed. Since the proposed MapReduce solution presented in this paper does not change the modeler outcome, demonstrating simulation results is out of the scope of this paper.

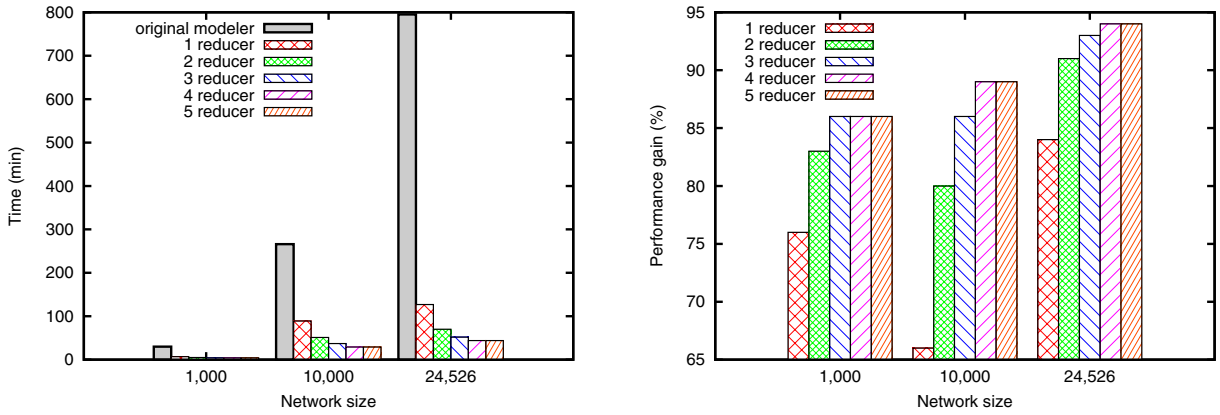
4 EXPERIMENTAL EVALUATION

We now describe an experimental evaluation of our proposed solution. Our proposed approach was initially done with the IBM InfoSphere BigInsights Enterprise Edition 2.0. We used the default configuration on a single node cluster with 8GB of RAM and 4 cores. This experiment was set up with the number of mappers corresponding to the number of input splits and a single reducer.

Tuning Hadoop and MapReduce to obtain optimum performance is not a trivial task. Most configuration settings are better tuned depending on the job behavior regarding CPU and memory usage, alongside with the cluster capacity. The number of mappers is typically controlled by the number of split size. Starting and stopping each map task can cause some overhead to the Hadoop performance. Therefore, it is not recommended to have mappers running only for a few seconds. One solution to this problem is to increase the block size, so that less map tasks are created. The number of reducers is controlled by the `mapred.reduce.tasks` configuration properties and it is crucial for performance if all of them run in only one wave. Hence, the number of reduce tasks must be greater or equal than the number of reduce slots in the cluster.

The number of map and reduce slots are defined by the `mapred.tasktracker.map.tasks.maximum` and `mapred.tasktracker.reduce.tasks.maximum` properties. These settings determine the maximum number of concurrently occupied slots on each `TaskTracker`, and the value needs to be based on the available CPUs and memory in the cluster. It is important to mention that hardcoding these values can have unintended consequences if the `TaskTrackers` does not have the same configuration. The number of slots has to be configured alongside with the heap sizes of mappers and reducers. We experimented with 1,024MB of heap size for map and reduce tasks and setting the maximum number of reducers slots and tasks to 6 caused an *OutOfMemory* error due to lack of resources. Also, other processes that run on the `TaskTracker` nodes should be considered, e.g., other components in the Hadoop distribution that the user is working with.

To improve the performance of our proposed solution we increased the number of reduce slots and tasks to 5. Figures 3(a) and 3(b) show the results with different configurations regarding number of reduce tasks while increasing the network size. In Figure 3(a) we can see how the original modeler linearly scales for larger network sizes. As for our proposed MapReduce solution the running time is sublinear when increasing the network size. We can also notice that increasing the number of reducer decreases the running time, but there is no performance improvement for more than 4 reducers. This indicates that for our experimental setup the optimal number of reducers is 4.



(a) Running time (min) *vs.* network size for different number of reducers.

(b) Performance gain (in %) *vs.* network size for different number of reducers.

Figure 3: Experimental results.

In Figure 3(b) we can see that our proposed solution with more than 3 reducers has a gain in performance up to 90% for network size 24,526 compared with the in memory-based solution. We can also observe for one and two reducers there is a decrease in performance gain from network size of 1,000 to 10,000 users. Also from four to five reducers there is no gain in performance due to some overhead in the reduce phase.

In summary, our proposed solution is highly scalable, with a sublinear behavior when increasing the network size. For our experimental environment the optimal number of reducers is 4, and there is no increasing for higher number of reducers.

5 RELATED WORK

Since the first seminal paper about the MapReduce framework, there are many other works proposing its use for a wide range of problems. Among them we can cite: the Asterix framework for handling and querying large XML data (Behm et al. 2011); set and all-pair similarity joins (Metwally and Faloutsos 2012); clustering large datasets (Kwon et al. 2010).

Common to all of the above works, the main purpose is to scale out data analytics workloads in cheap clusters. Experimental evaluations comparing different approaches with the main goal of observing the behavior of MapReduce with respect to scaling out and scaling up properties have been presented in (Pavlo et al. 2009; Appuswamy et al. 2013; Sevilla et al. 2013).

Some research in this domain relate closely to this work. In Pujol et al. (2010), the problem of scaling OSN is tackled by a partitioning and replication middleware solution that takes advantage of the network structure to achieve data locality while minimizing replication. In Tang et al. (2009), the use of the MapReduce framework is studied to scale the problem of modeling the topic-level social influence on large networks. In that work, topic-level influence propagation is performed given the results of topic modeling and the network structure.

Nevertheless, our proposed solution differs from all the above works since we propose a method to scale multi-agent based simulation of large social networks. To the best of our knowledge, this paper is the first to tackle the problem of simulation of agent-based information diffusion models on large OSN using the MapReduce framework.

6 CONCLUSIONS AND FUTURE WORK

This paper is the first of its kind to tackle the problem of building agent- based models using massive amount of data. Our main goal is running simulations of agent-based information diffusion models on large

OSN. Our proposed MapReduce solution was implemented and tested with real OSN data. Experimental results showed the efficiency and scalability of our proposed solution. An important experimental finding of our solution is that the performance gain can be up to 90% when increasing the network size. Moreover, our proposed solution has a sublinear running time when increasing the network size. As a future research we plan to evolve the modeler to build more complex behavior and to perform more experiments on larger OSN samples.

REFERENCES

Appuswamy, R., C. Gkantsidis, D. Narayanan, O. Hodson, and A. Rowstron. 2013. "Scale-up vs Scale-out for Hadoop: Time to rethink?". In *Proceedings of the 4th Annual Symposium on Cloud Computing*, Article 20. ACM.

Behm, A., V. R. Borkar, M. J. Carey, R. Grover, C. Li, N. Onose, R. Vernica, A. Deutsch, Y. Papakonstantinou, and V. J. Tsotras. 2011. "ASTERIX: towards a scalable, semistructured data platform for evolving-world models". *Distributed and Parallel Databases* 29 (3): 185–216.

Cha, M., A. Mislove, and K. P. Gummadi. 2009. "A measurement-driven analysis of information propagation in the Flickr social network". In *Proceedings of the 18th International Conference on World Wide Web*, 721–730.

Dean, J., and S. Ghemawat. 2008. "MapReduce: Simplified Data Processing on Large Clusters". *Communications of the ACM* 51 (1): 107–113.

DeWitt, D., and J. Gray. 1992. "Parallel Database Systems: The Future of High Performance Database Systems". *Communications of the ACM* 35 (6): 85–98.

Gatti, M. A. C., A. P. Appel, C. N. Santos, C. S. Pinhanez, P. R. Cavalin, and S. M. B. Neto. 2013. "A Simulation-based Approach to Analyze the Information Diffusion in Microblogging Online Social Network". In *Proceedings of the 2013 Winter Simulation Conference*, edited by R. Pasupathy, S.-H. Kim, A. Tolk, R. Hill, and M. E. Kuhl, 1685–1696. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Gatti, M. A. C., A. P. Appel, C. N. Santos, C. S. Pinhanez, P. R. Cavalin, and S. M. B. Neto. 2014. "Large-Scale Multi-Agent-based Modeling and Simulation of Microblogging-based Online Social Network". In *Multi-Agent-Based Simulation XIV*, edited by S. J. Alam and H. V. D. Parunak, 17–33: Springer.

Gruhl, D., R. Guha, D. Liben-Nowell, and A. Tomkins. 2004. "Information diffusion through blogspace". In *Proceedings of the 13th International Conference on World Wide Web*, 491–501.

Jennings, N. R. 2001. "An agent-based approach for building complex software systems". *Communications of the ACM* 44 (4): 35–41.

Kempe, D., J. Kleinberg, and E. Tardos. 2003. "Maximizing the spread of influence through a social network". In *Proceedings of the 9th ACM International Conference on Knowledge Discovery and Data Mining*, 137–146.

Kwon, Y., D. Nunley, J. P. Gardner, M. Balazinska, B. Howe, and S. Loebman. 2010. "Scalable Clustering Algorithm for N-Body Simulations in a Shared-Nothing Cluster". In *Scientific and Statistical Database Management*, edited by M. Gertz and B. Ludäscher, 132–150. Springer.

Leskovec, J., L. A. Adamic, and B. A. Huberman. 2006. "The dynamics of viral marketing". In *Proceedings of the 7th ACM Conference on Electronic Commerce*, 228–237.

Leskovec, J., and E. Horvitz. "Planetary-scale views on a large instant-messaging network". In *Proceedings of the 17th International Conference on World Wide Web*.

Metwally, A., and C. Faloutsos. 2012. "V-SMART-Join: A scalable mapreduce framework for all-pair similarity joins of multisets and vectors". *Proceedings of the VLDB Endowment* 5 (8): 704–715.

Neto, S. M. B., M. A. C. Gatti, P. R. Cavalin, C. S. Pinhanez, C. N. Santos, and A. P. Appel. 2013. "Reaction Times for User Behavior Models in Microblogging Online Social Networks". In *Proceedings of the 2013 Workshop on Data-Driven User Behavioral Modelling and Mining From Social Media*, 17–20.

Pavlo, A., E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. Madden, and M. Stonebraker. 2009. "A Comparison of Approaches to Large-scale Data Analysis". In *Proceedings of the 2009 ACM International Conference on Management of Data*, 165–178.

Pujol, J. M., V. Erramilli, G. Siganos, X. Yang, N. Laoutaris, P. Chhabra, and P. Rodriguez. 2010. "The Little Engine(s) That Could: Scaling Online Social Networks". *ACM SIGCOMM Computer Communication Review* 40 (4): 375–386.

Rogers, E. M., and E. Rogers. 2003. *Diffusion of Innovations*. 5th ed. Free Press.

Sevilla, M., I. Nassi, K. Ioannidou, S. Brandt, and C. Maltzahn. 2013. "A Framework for an In-depth Comparison of Scale-up and Scale-out". In *Proceedings of the 2013 International Workshop on Data-Intensive Scalable Computing Systems*, 13–18.

Strang, D., and S. A. Soule. 1998. "Diffusion in Organizations and Social Movements: From Hybrid Corn to Poison Pills". *Annual Review of Sociology* 24 (1): 265–290.

Tang, J., J. Sun, C. Wang, and Z. Yang. 2009. "Social Influence Analysis in Large-scale Networks". In *Proceedings of the 15th ACM International Conference on Knowledge Discovery and Data Mining*, 807–816.

AUTHOR BIOGRAPHIES

MAÍRA A. C. GATTI is a Research Staff Member in the Social Data Analytics Group at IBM Research-Brazil. Her main area of expertise is in Computer Science and her interests are in Distributed Systems and Multi-Agent-based Simulation. Her email address is mairacg@br.ibm.com.

MARCOS R. VIEIRA is a Research Staff Member in the Natural Resource Analytics Group at IBM Research-Brazil. His main areas of interests are in Data Analytics and Large Scale Systems. His email address is mvieira@br.ibm.com.

JOÃO PAULO F. DE MELO is an undergraduate student at Universidade Federal Fluminense and is working as a research intern at IBM Research-Brazil. His email address is jforny@br.ibm.com.

PAULO RODRIGO CAVALIN is a Research Staff Member in the Social Data Analytics Group at IBM Research-Brazil. His main research fields are Pattern Recognition and Machine Learning. His email address is pcavalin@br.ibm.com.

CLAUDIO SANTOS PINHANEZ leads the Social Data Analytics Group at IBM Research-Brazil, working on Social Computing, Service Science and Human-Computer Interfaces. Claudio got his Ph.D. in 1999 from the MIT Media Laboratory. His email address is csantosp@br.ibm.com.