Proceedings of the 2018 Winter Simulation Conference
M. Rabe, A. A. Juan, N. Mustafee, A. Skoogh, S. Jain, and B. Johansson, eds.

A PREDICTIVE SELF-CONFIGURING SIMULATOR FOR ONLINE MEDIA

Tarek Abdelzaher, Yifan Hao, Dongxin Liu,
Shengzhong Liu, Huajie Shao, Shuochao Yao

Department of Computer Science
University of Illinois
Urbana, IL 61801

James Flamino[1], Boleslaw Szymanski[2]

[1]Department of Physics
[2]Department of Computer Science
Rensselaer Polytechnic Institute
Troy, NY 12180

ABSTRACT

This paper describes the design, implementation, and early experiences with a novel agent-based simulator of online media streams, developed under DARPA's SocialSim Program to extract and predict trends in information dissemination on online media. A hallmark of the simulator is its self-configuring property. Instead of requiring initial set-up, the input to the simulator constitutes data traces collected from the medium to be simulated. The simulator automatically learns from the data such elements as the number of agents involved, the number of objects involved, and the rate of introduction of new agents and objects. It also develops behavior models of simulated agents and objects, and their dependencies. These models are then used to run simulations allowing future extrapolation and "what if" analysis. Results are presented on using this system to simulate GitHub transactions. They show good performance in terms of both simulation accuracy and overhead.

1 INTRODUCTION

In the 20th century, advances in computing enabled significant advances in the understanding and exploitation of physical systems thanks to faster modeling and simulation capabilities. The 21st century is widely believed to be the age of *information*. By interconnecting individuals and endowing them with unprecedented capabilities for sharing information at scale, a new global information fabric has emerged, wherein information signals are created, propagated, distorted, and consumed, not unlike signals on physical media. Yet, the properties of this new information fabric remain poorly understood.

Can advances in cyber-physical computing help uncover the properties of the new information fabric, the way they helped study laws of physical signals, thereby laying the analytic foundations for understanding online information dynamics in the 21st century? Information objects on online media are diverse. They include posts on social networks, such as Twitter, publication entries in databases, such as DBLP, and software updates on version control services, such as GitHub. Collectively, streams of these continually created and updated objects comprise a new reflection of reality/beliefs (e.g., on Twitter), innovation (e.g., on GitHub), and publications (e.g., on DBLP). What are the fundamental properties of this reflection? Does it reveal large scale phenomena such as competition (e.g., between ideas/beliefs, software toolchains, or publication pedigrees)? Does it predict the rise and fall in popularity of the underlying objects involved? Does it help understand synergies between ideologies, research directions, or innovations? How does the landscape of synergies and competitions mold the dynamics of online interactions?

A first step towards answering those questions is to develop good simulation tools of information dissemination on online media. Those tools will advance scientific discovery towards addressing the above questions, much the way wind tunnel simulators helped create more efficient aircraft, and data center simulators helped inform energy management policies.

This paper describes a new simulator of online media. The simulator has two innovative features. First, it is highly retargetable. It can be adapted to simulate multiple different online media that share the abstractions of agents, (information) objects, and actions. Second, it is self-configuring. While a user can decide the configuration being simulated, it is possible to generate the configuration automatically from past traces of the target online medium.

The rest of the paper is organized as follows. Section 2 summarizes the overall architecture of the simulator. Section 3 describes the modules responsible for agent simulation and clustering. Section 4 presents initial evaluation results. Related work is reviewed in Section 5. The paper concludes with Section 6.

2 ARCHITECTURE

Our online media simulations rely on three main abstractions: *agents*, information *objects*, and *actions*. Agents perform actions on information objects. These actions may have dependencies on other actions, which is how signals spread through the medium. For example, on Twitter, tweets constitute the information objects. Users constitute agents. Individual tweets can be created or re-posted (retweeted), which constitutes actions. On GitHub, repositories are information objects. GitHub users are agents. Users can create a repository, fork it, commit content to it, or express interest in it ("star" it), among other capabilities that constitute actions.

2.1 Design Objectives

Our simulation architecture supports two novel features; namely, being retargetable and being self-configuring. These features are introduced below:

- **Retargetable simulation:** Our goal is to simulate multiple types of online media, centered around the abstractions of *agents*, *objects*, and *actions*. As described later in this paper, simulated configurations can be instantiated from data learned from media traces. To make the simulator retargetable, it is therefore important to convert media traces into a unified set of supported formats. This is the responsibility of a preprocessing step. Presently, it converts media-specific data structures into both a MySQL database and JSON files. Later modules can thus pick their preferred input format independently of the original source.
- **Self-configuring simulation:** Our online media simulator is novel in that it learns its own agent and object configuration and models from traces of the target online medium. An agent in the simulator is defined by a model that features several parameters. These parameters are learned by the *analysis library*. This library reads traces of online media (from either the MySQL database or the JSON files mentioned above) and computes statistics from which agent model parameters can be determined. Developers are free to implement and experiment with different kinds of machine learning or data mining algorithms, and integrate them into the analysis library to extract behaviors of different agents from the input data. These statistics constitute the *configuration file* needed for the simulator to operate.

When the simulator starts, an agent builder module reads the configuration files using a proxy that converts the stored configuration into a run-time instantiation, of agent models. The simulator core module then runs the agent models and logs the observed behaviors.

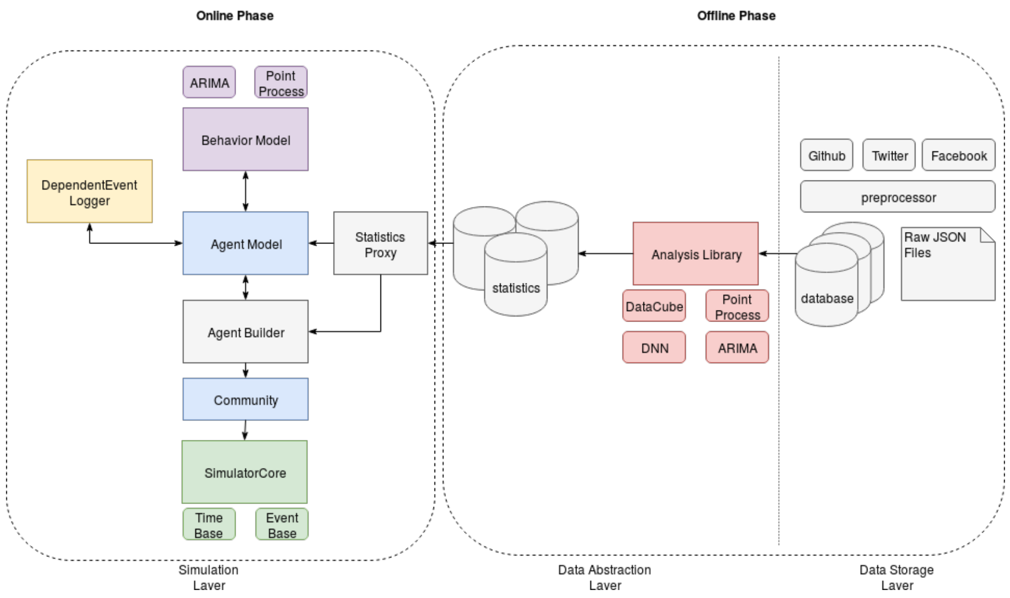


Figure 1: SocialCube simulator architecture

2.2 Simulator Architecture

The simulator architecture is shown in Figure 1. Once a configuration file has been generated, the simulation can start. The *statistics proxy* in Figure 1 serves as a bridge between the offline phase and the online phase. The proxy needs to understand the format of different analysis outputs. A new statistics proxy module needs to be integrated whenever a new type of analysis is implemented in the library. The architecture makes it easy to extend the simulator with new types of analysis techniques.

The *behavior model* is a collection of algorithms describing how the agent parameters described in the configuration file determine agent behavior. It is a place to simulate the behaviors and actions of an agent. A new behavior model is also needed whenever a new type of analysis (and hence new agent parameters) are introduced.

The *agent model* is a class that implements individual agents. Each agent will encapsulate a certain behavior model. Each agent (instance of the agent model class) also encapsulates the variables that are specific to the agent. It is essentially a stateful wrapper around the behavior model, whereas the behavior model describes the algorithm that interprets the variables.

The *agent builder* is a factory that instantiates agents during a simulation. When the simulator starts, the agent builder gets a list of agents from the statistics proxy. It then instantiates all the agents of the appropriate type and behavior model. The agent builder also creates new users on the fly according to a pattern dictated in the configuration file.

Note that, some agents represent communities. Like normal agents, they have behavior models, except that the generated actions do not share a single user ID, but rather are distributed across multiple identities. Communities can be used to cluster similar agents that share the same behavior model parameters.

The *event logger* records key past events in the simulation. Developers can adjust the depth of the logger, which in turn will adjust the memory of the simulation. Agent behavior models can condition behaviors based on events logged.

The *simulator core* will hold all the agents and communities after the agent builder instantiates them. The simulator core manages the passage of time, making calls to agents (with reference to current time and logged events) so they can be concurrently simulated.

3 AGENT MODELS AND SIMULATION ALGORITHMS

The simulator offers duality between agents and objects. In that sense, it can either run in an *agent-based* mode, or *object-based* mode. In the agent-based mode, the main abstraction of the simulator is the agent. The core simulation loop executes agents in (simulated) concurrency, one time-slot at a time. The model of each simulated agent is executed, producing the actions the agent will perform on objects during the simulated time interval. In the object-based mode, the main abstraction of the simulator is the object. The model of each simulated object is executed, one time slot at a time, producing the actions that agents will be perform on the object during the simulated time interval.

The duality between agents and objects means that the two simulation modes and underlying data structures are symmetric. Below, we explain the data structures used in the agent-based mode. The data structures used in the object-based mode are symmetric and can be described simply by replacing "agent" with "object" and vice versa in the description below.

In the agent-based mode, an agent model is described by (i) total *activity rate* (i.e., rate of actions), (ii) *affinities* (set of objects it interacts with), and (iii) *interaction rates* (frequency of interaction with each object). The total activity rate can also be broken by action type (e.g., tweets versus retweets). A Twitter user might, for example, have different rates for original tweets versus re-tweets. Users who represent official entities such as local traffic alerts might post predominantly original tweets. In contrast, propaganda publicists might predominantly curate and re-post previously published news that favor their viewpoints, thus primarily posting retweets.

Finally, the above parameters are recomputed at multiple time-scales. The current architecture supports two time-scales for parameter re-computation, called *global* and *local*, respectively. The global time-scale is slower. It adjusts aggregate parameters such as total activity rate and affinities by extrapolating trends described by auto-regressive time-series models. The local time-scale is shorter. It adjusts changes in probabilities of individual action types by extrapolating a more detailed point process model. Below, we describe these models in more detail.

3.1 Long-term Agent Trends

We predict the long-term trends (*e.g.*, trends in total activity rate and affinities of agents) by applying time-series analysis models to extrapolate from observations. Here, we use the autoregressive integrated moving average (ARIMA) for the extrapolation.

We consider the aggregate parameters for each agent (or group of agents) a time series, and use X_t to denote the data vector at time slot t, where t is an integer. The ARIMA model can be expressed by "ARIMA(p,d,q)", where p refers to the autoregressive terms, d refers to the order of differences in the model, and q refers to white noise terms.

We first introduce the difference (Hamilton 1994) of a time series. Let X_t^d denote the d^{th} order difference of X_t:

$$X_t^d = \begin{cases} X_t, & \text{if } d = 0, \\ X_t^{d-1} - X_{t-1}^{d-1}, & \text{if } d \geq 1. \end{cases} \tag{1}$$

Then, the ARIMA(p,d,q) model can be written as:

$$X_t^d - \sum_{i=1}^{p} \phi_i X_{t-i}^d = c + \varepsilon_t - \sum_{j=1}^{q} \theta_j \varepsilon_{t-j} \tag{2}$$

where c is a constant, the sequence $\{\varepsilon_t\}$ consists of independently identically distributed (iid) Gaussian noise terms with mean zero and variance σ^2, and ϕ_i and θ_j refer to the coefficients of X_{t-i}^d and ε_{t-j}, respectively.

The parameters of ARIMA can be learned by maximizing likelihood. Assuming, without loss of generality, that $c = 0$, the likelihood function for the parameters $\phi_1, \phi_2, \ldots, \phi_p$, $\theta_1, \theta_2, \ldots, \theta_q$ and σ is:

$$L(\phi, \theta, \sigma | X) = \frac{1}{\sqrt{2\pi\sigma^2|\Sigma|^2}} \exp(-\frac{1}{2\sigma^2} x^T \Sigma^{-1} x), \tag{3}$$

where $\phi = [\phi_1, \phi_2, \ldots, \phi_p]^T$, $\theta = [\theta_1, \theta_2, \ldots, \theta_q]^T$, $x = [X_1^d, X_2^d, \ldots, X_n^d]^T$, $\Sigma = \sigma^{-2}\mathrm{cov}(x)$, and n is the number of the training data samples. The exact form of $L(\phi, \theta, \sigma|X)$ is too complicated to maximize directly. Hence, several approximations and modifications were proposed (Anderson 1977; Shaman 1975; Ljung and Box 1979). After getting the parameters of the ARIMA model, we can predict the value of X_t^d using Equation (2).

For each agent (or group of agents), the analysis library builds an ARIMA model to predict the rate of each type of action and the interaction rates with individual objects over time.

3.2 Agent-Agent Influence

In addition to predicting long-term agent behavior trends, we also estimate influence between different agents, expressed by directional dependencies between their actions on shared objects. Here, we use the temporal point process model, specifically the Hawkes process model, which is proposed by Hawkes (Hawkes 1971) to describe dependencies across different event types. Point processes provide the statistical language to describe the timing and properties of events (Rizoiu et al. 2017). They proved to be successful in many fields, including financial applications (Embrechts et al. 2011), geophysics analysis (Ogata 1988), and information propagation on online social media (Zhao et al. 2015).

The Hawkes process is a natural extension of the well-known Poisson process. Instead of assuming that events happen independently as in a Poisson process, the probability of each coming event is dependent on the every past event in the Hawkes process (with a decay function that effectively results in finite memory). A Hawkes process sequence can be completely specified by the distribution of its inter-event times:

$$f(t_1, \ldots, t_n) = \prod_{k=1}^{n} f(t_k | t_1, \ldots, t_{k-1}) = \prod_{k=1}^{n} f(t_k | \mathscr{H}_{t_k}). \tag{4}$$

where $\mathscr{H}(t)$ denotes the event history before time t and $f(t_k)$ denotes the probability density function (PDF) of the kth event instant. To derive $f(t|\mathscr{H}(t))$, we need to define the conditional intensity $\lambda(t)$ first, which is heuristically defined as the arrival rate of new events:

$$\lambda = \lim_{\Delta t \to 0} \frac{\mathbb{E}(N(t + \Delta t) - N(t)|\mathscr{H}_t)}{\Delta t} = \lim_{\Delta t \to 0} \frac{\mathbb{P}(N(t + \Delta t) - N(t) = 1)|\mathscr{H}_t)}{\Delta t}. \tag{5}$$

where, $N(t)$ is the number of events up to time t. The relationship between the conditional intensity function $\lambda(t)$ and the probability density function $f(t|\mathscr{H}_t)$ is given by:

$$\lambda(t) = \frac{f(t|\mathscr{H}_t)}{1 - F(t)}. \tag{6}$$

where $F(t)$ is the cumulative distribution function (CDF) of the given event sequence up to time t. It is also the integral of $f(t|\mathscr{H}_t)$ from the start to time t. Therefore, the conditional intensity function is all we need in order to define a Hawkes process.

Now let's come back to our scenario. We want to use the point process model to describe the dependencies between events of different types. We model it from an object-centric viewpoint, because on online media the operations by different agents on the same object are usually more interrelated than operations by the same agent on different objects. For example, agents may influence other agents to retweet their text, cite their paper, or view their repository update. The object-centric view-point, therefore, captures influence among agents. For each dimension (event type), i, the conditional intensity function is:

$$\lambda_i(t) = \mu_i + \sum_{j=1}^{D} \sum_{t_k^j < t} \phi_{ji}(t - t_k^j), \tag{7}$$

where the triggering kernel $\phi_{ji}(t)$ from event type j to event type i is an exponential decay function with respect to the time elapsed:

$$\phi_{ji}(t) = \alpha^{ji} \beta^i \exp(-\beta^i t) 1_{t>0}. \tag{8}$$

In this definition, for an object with L event types, we have three parts of parameters in total:

- μ: a vector of size L. It represents the base intensity of each dimension. It determines the arrival rate of events triggered by extraneous factors.
- α: a matrix of shape $L \times L$. It denotes the triggering coefficients within and across each dimension.
- β: a vector of size L. It represents the decay factor of the triggering effect. We have to set the triggering effect between dimensions as a time decay function, otherwise it will result in an infinite loop of event triggering.

With the definition of our object Hawkes process model, we are able to derive the log likelihood function of seeing the observed event time series. We maximize this function with respect to model parameters using the Broyden-Fletcher-Glodfarb-Shanno(BFGS) algorithm (Liu and Nocedal 1989), a widely used quasi-Newtons method for unconstrained optimization problems without requiring the computation of a second-order Hessian matrix. The resulting mode is used to predict future events using Equation (4).

When both the ARIMA and the point process model are jointly used, we use the former to define long-term trends in total action budgets, and the latter to define action timing within those budgets.

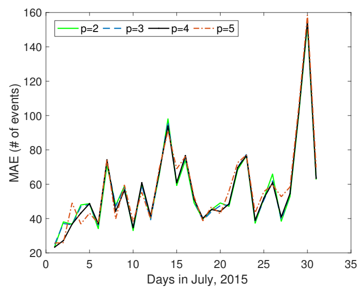
3.3 Communities and Aggregation

Aggregation is a central component for agent-based simulations, as this allows for grouping of both agents and objects by similarity in behavior, which naturally reduces overhead at runtime and can even capture behavior of communities more accurately, if insufficient data is present on individual members.

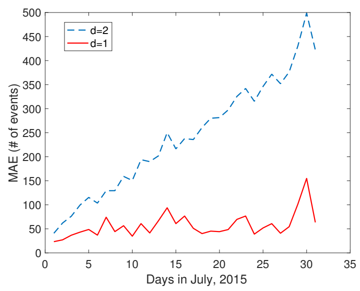
One type of aggregation is the grouping of agents by similarity in temporal behavior. For example, clear daily cycles exist for a vast majority of users that are well-correlated with their time zones. Some networks, such as GitHub, also see a significant drop-off of activity on weekdays. The concept of agent clustering is not limited, of course, to time zone effects. The simulator leaves it open to define meta-agents that correspond to clusters of agents with similar behaviors. Ideally, such meta agents are used to model communities of agents when one does not have a sufficient number of data points to derive parameters of their individual agent model (e.g., per-agent ARIMA model parameters). If agents are grouped by predicted similarity, the group can collectively have enough data to be modeled accurately. The prediction of similarity is up to individual routines in the analysis library. For example, it may be based on organization, role, time-zone, or a combination thereof.

4 EVALUATION

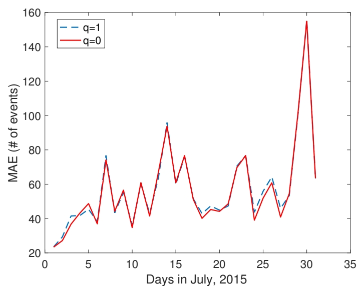
In this section, we present some of the basic results of running components of the analysis library and simulation on data obtained from GitHub. The dataset used in this section is provided by Leidos. It covers



(a) Different p for ARIMA for PushEvent

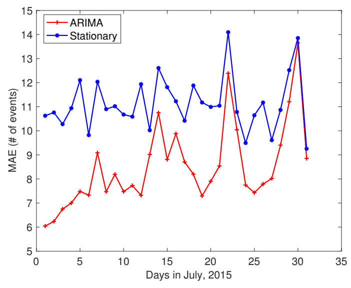


(b) Different d for ARIMA for PushEvent

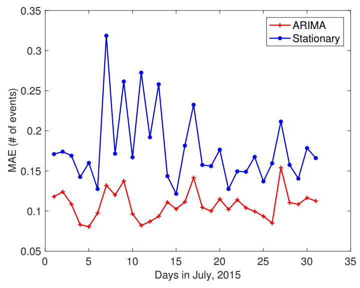


(c) Different q for ARIMA for PushEvent

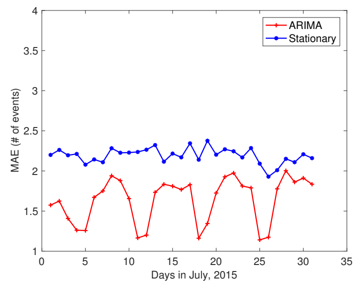
Figure 2: Parameters tuning for ARIMA model



(a) Comparison of MAE for Push event



(b) Comparison of MAE for Fork event



(c) Comparison of MAE for PullRequest event

Figure 3: Comparison of MAE for ARIMA and stationary

records from Jan 2015 to Aug 2017. All the records have been anonymized to protect privacy of Github users. Each record specifies information of actor (user agent), payload, timestamp, action type, as well as repository id (object id).

4.1 Predicting Activity Rate and Type

In this experiment, we are going to predict the number of different event types for a test period (month) for each user on Github. On Github, there are about 10 different event types such as PushEvent, ForkEvent, DeleteEvent and PullRequest. Due to space limitations, we use only the three more common event types, PushEvent, ForkEvent and PullRequest, as representatives. We compare ARIMA and a stationary method (where the future prediction is set identical to the current rates). First, we investigate the impact of model structure (i.e., parameters, p, d, and q) on accuracy of the ARIMA model. We use the data from January to June in 2015 to train the ARIMA model, then test its performance on the data in July 2015. Fig. 2 shows the mean absolute error (MAE) for different values of parameters p, d, q in predicting PushEvent rates. From these (and similar experiments), we see that $p = 4$, $d = 1$, and $q = 0$ give best accuracy.

The next step is to predict the number of event types for each user using ARIMA model. It took about 2 hours to train the analysis library to generate models of 2000 agents based on six months of training data. Fig. 3 shows the mean absolute error (MAE) in predicting the rate of different event types of the top 2000 users using ARIMA and the stationary method (where future is equal to the present). As expected, we observe that the ARIMA method outperforms the stationary method.

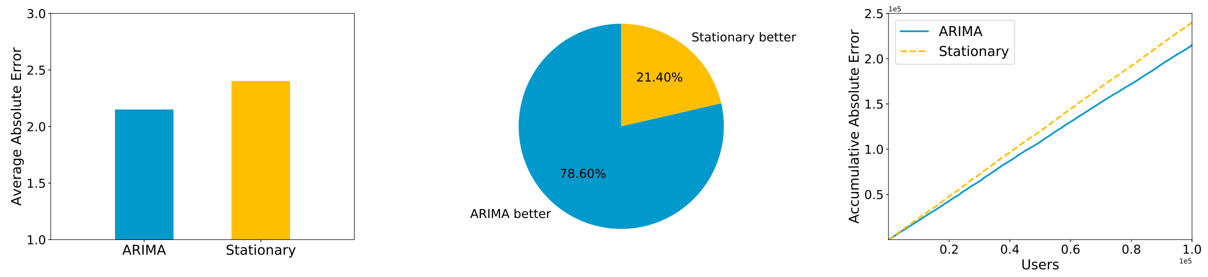
After predicting the number of different event types for each user, we can compute the probability distribution of each event type. We take one user as an example in Table 1. The second row is the number of events predicted by the ARIMA model for this user while the last row is the corresponding probability distribution. This probability distribution would then be part of the configuration file for simulating the desired epoch of the user.

Table 1: Probability distribution of different events for a certain user

Issues	PullReviewComment	Push	PullRequest	IssueComment	Create	Watch	Fork	Delete	CommitComment
95	197	3796	45	893	25	0	0	4	0
.0188	.0389	.7509	.0089	.1766	.0049	0	0	.0008	0

4.2 Predicting Affinity

In this part, we evaluate our ARIMA model for predicting affinities of the agents, and then compare the performance with the stationary agent model. We use all the Github events from January to June in 2015 to train our ARIMA model and use the data in July as the test set. Per affinity definition in Section 3, for Github, affinities refers to the set of repositories a user contributes to. In the evaluation, we build the agent model for 10^5 users who are active in June 2015, and predict their interaction rates with their repositories in July 2015 based on our ARIMA model. We then compare to the stationary model. We use the difference between the ground truth and the prediction values as the absolute errors, and define the accumulated error as the accumulation of the absolute errors over users. We run our ARIMA model using two cores of an Intel i7 CPU. It took less than one hour to train the analysis library to generate models of 10^5 agents based on the 6 months of training data.



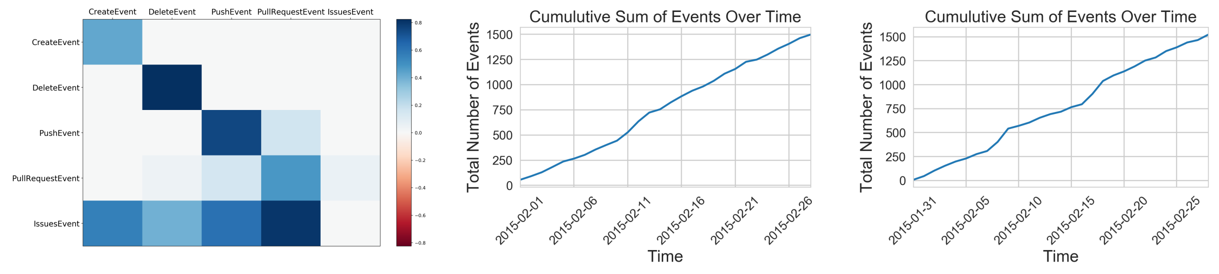
(a) Average Absolute Error (b) Ratio of Smaller Absolute Errors (c) Accumulative Avsolute Error

Figure 4: Comparison between ARIMA and stationary

Figure 4(a) shows the average absolute error of ARIMA and the Stationary model over the 10^5 users. We observe that our ARIMA model generates a smaller absolute error overall. Breaking it down by user, Figure 4(b) shows that ARIMA gives better error for 78.6% users, while the Stationary model performs better for the remaining 21.4%). Figure 4(c) shows the cumulative absolute errors, verifying the advantages of ARIMA-based prediction.

4.3 Understanding Short-term Dependencies

In this subsection, we briefly present the evaluation results for simulated events provided by the short-term point process model we introduced before. First, we describe the performance of single repository simulation results. Second, we enlarge the scope of comparison and evaluate aggregate performance on a set of popular repositories.



(a) Triggering Kernels (b) Groundtruth Events (c) Simulation Events

Figure 5: Estimation and simulation results on repository ”JwcbE-owDa9ymJm3D4xhsw”.

In the first experiment, we randomly choose one repository out of the January 2015 event set and estimate its point process parameters. Then we simulate this model in February 2015 and compare it with ground truth. The estimated triggering kernels, and the cumulative distributions of number of events over time are shown in Figure 5. From the triggering kernel map, we can see that the different event types on this repository are relatively independent because the triggering coefficients are higher in diagonal elements except for the IssuesEvent. As presented in this figure, our point process model maintains the daily activity level (i.e., slope of curve) and the burstiness (i.e., smoothness of curve) well in the simulation.

In addition, we compare other statistical metrics in the Table 2. In this table, the diffusion delay is defined as the time difference between the time of first event on this repository and the time of each following event. It represents the average offset of all the events in this repository. Our simulation is within 10% error of the groundtruth. The simulation also performs well on the average inter-event time and total number of events because it is designed to model the triggering effects on inter-event times.

Table 2: Comparison on single repository.

Metric	Groundtruth	Simulation
Average Diffusion Delay	325.599 h	348.077 h
Average Inter-event Time	1575.643 s	1579.644 s
Total Number of Events	1497	1532

Next, to test the performance of our model on more data, we choose the 1000 most popular repositories in the first half year of 2015 to perform simulation. In this experiment, we only show the WatchEvent and ForkEvent, which reflect the popularity of these repositories. The training period is January 2015 to June 2015, and the simulation is conducted in July 2015. We first independently estimate the parameters for each repository, then perform separate simulation on them, and finally combine the simulated events together to compare with the groundtruth events on these repositories. The total training time of the analysis library for the 1000 repositories is about 6 hours on a server with 48 Intel Xeon CPUs@2.50GHz and 256 GB memory, where the total number of events on these repos is beyond 1.4 million. However, the simulation of these repositories will only last for 250 $\sim$ 300 seconds to generate one month of results, running on a single core. The comparison results of accuracy are given in Table 3.

Table 3: Comparison on top 1000 repositories.

Metric	Groundtruth	Simulation
Average Diffusion Delay	372.401191 h	373.485502 h
Average Inter-event Time	83815.010660 s	80263.662305 s
Total Number of Events	242247	275133

In Table 3, since we include more repositories in the comparison, the diffusion delay distribution is more balanced, resulting the average delay very close to the middle point of a month. We still obtain good accuracy on average inter-event time (relative error $< 5\%$) and total number of events (relative error $\simeq 12\%$), which proves the stability of the point process model.

4.4 Trend Aggregation

Here, we evaluate both the efficiency and accuracy of clustering by temporal behavior patterns. In the simulation, the offline analysis library is fed a data set with a date range of 2016-03-01 to 2016-04-01, comprising 27803145 individual events. We compute regional predictions of temporal activity patterns and use them to eliminate agents that are (predicted to be) asleep from simulation at every time step. The results are detailed in Table 4.

Table 4: Evaluation results for temporal preference clustering

	With Activity Level Prediction	Full Simulation
Runtime (sec)	42.5	203.3
RMSE	305.4	273.1

The table shows that eliminating users predicted to be inactive based on time zone saves 79.1% of simulation time, while the percent difference in simulation error (RMSE) in predicting hourly PushEvents for the simulated month is only 11.2%. Better clustering remains an open challenge and an active area of development of analysis libraries for our simulator. Regardless, the decrease in overhead is substantial and the speed-up scales well with the number of agents in the training data.

5 RELATED WORK

A key direction in simulation literature is to empower diverse application domains with agent-based simulators for in-depth understanding of real-world complexities, and forecasting quantities of interest. Railsback et al. proposed an agent-based simulation platform for development recommendations (Railsback et al. 2006). Drogoul *et al.* design a multi-agent-based simulator for distributed artificial intelligence research (Drogoul et al. 2002). Bunn *et al.* use agent-based simulation for electricity trading arrangements of England and Wales (Bunn and Oliveira 2001). Raberto *et al.* apply agent-based simulation for understanding the financial markets (Raberto et al. 2001).

Recently, simulations for online media have attracted a growing interest from the community. Abundant studies have been made for understanding the influence of social networks on human daily lives and decision making (Zhang et al. 2015; Lanham et al. 2014), validating online viral marketing (Serrano and Iglesias 2016; Sela et al. 2016), and modelling rumor propagation (Serrano et al. 2015; Kaligotla et al. 2015). One key link missing from the picture, however, is a flexible and self-configuring simulation framework that integrates various online media with a modular interface designed for extensibility of data analytics and prediction functions.

With the proliferation of interconnected users, many studies were made on modelling user behaviour. Wang *et al.* jointly model user and object interactions with latent credible status and user dependencies (Wang et al. 2014; Wang et al. 2015). Yao *et al.* incorporate time information and compute a fundamental error bound (Yao et al. 2016; Yao et al. 2016). Shao *et al.* formulate a nonlinear integer programming problem to select optimal sources to solicit in order to minimize the expected fusion error (Shao et al. 2017). Zhao *et al.* focus on modeling the evolution of popularity on online media by mining big information cascades (Zhao et al. 2015). Hoang *et al.* study the online propagation behaviors of micro-blogging content (Hoang and Lim 2016). From the prospective of online media simulation, it is not easy to design plug-in modules that accommodate such diverse considerations. To the best of our knowledge, this paper develops a novel modular framework for integrating models at different scales to address difference micro/macro concerns while allowing self-configuration and easy retargeting.

The paper describes a novel highly retargetable simulator of online media. With systemically designed self-configuration modules, we make automatic learning and self-configuration of online media simulators accessible.

6 CONCLUSIONS

In this paper, we described a new simulator of social media streams. The simulator is novel in that it does not require manual configuration. Rather, the simulated medium is automatically instantiated from analysis of past data traces from the target medium. Prediction algorithms allow extrapolating from the past trace to a predicted future. Preliminary evaluation results show that the predictions are accurate and scalable. The simulator is extensible. As better prediction and clustering algorithms are developed over time, they can be integrated as modules in the analysis library and as agent behavior models. Similarly, as new social media are considered, they can be converted into an agent/object/action representation by new thin pre-processing modules, allowing existing analysis algorithms to extract parameters of agents and objects for the new medium. Thus, analysis and simulation code can be reused across different online media. Future work of the authors will address performance optimization of the simulator, extensions of analysis libraries and agent behavior models, and support for additional input media types.

ACKNOWLEDGMENTS

This work was sponsored in part by DARPA under contract W911NF-17-C-0099.

REFERENCES

Anderson, T. (1977). Estimation for autoregressive moving average models in the time and frequency domains. *The Annals of Statistics*, 842–865.

Bunn, D. W. and F. S. Oliveira (2001). Agent-based simulation-an application to the new electricity trading arrangements of england and wales. *IEEE transactions on Evolutionary Computation 5*(5), 493–503.

Drogoul, A., D. Vanbergue, and T. Meurisse (2002). Multi-agent based simulation: Where are the agents? In *International Workshop on Multi-Agent Systems and Agent-Based Simulation*, pp. 1–15. Springer.

Embrechts, P., T. Liniger, and L. Lin (2011). Multivariate hawkes processes: an application to financial data. *Journal of Applied Probability 48*(A), 367–378.

Hamilton, J. D. (1994). *Time series analysis*, Volume 2. Princeton university press Princeton.

Hawkes, A. G. (1971). Spectra of some self-exciting and mutually exciting point processes. *Biometrika 58*(1), 83–90.

Hoang, T.-A. and E.-P. Lim (2016). Microblogging content propagation modeling using topic-specific behavioral factors. *IEEE Transactions on Knowledge and Data Engineering 28*(9), 2407–2422.

Kaligotla, C., E. Yücesan, and S. E. Chick (2015). An agent based model of spread of competing rumors through online interactions on social media. In *Winter Simulation Conference (WSC), 2015*, pp. 3985–3996. IEEE.

Lanham, M. J., G. P. Morgan, and K. M. Carley (2014). Social network modeling and agent-based simulation in support of crisis de-escalation. *IEEE Transactions on Systems, Man, and Cybernetics: Systems 44*(1), 103–110.

Liu, D. C. and J. Nocedal (1989). On the limited memory bfgs method for large scale optimization. *Mathematical programming 45*(1-3), 503–528.

Ljung, G. M. and G. E. Box (1979). The likelihood function of stationary autoregressive-moving average models. *Biometrika 66*(2), 265–270.

Ogata, Y. (1988). Statistical models for earthquake occurrences and residual analysis for point processes. *Journal of the American Statistical association 83*(401), 9–27.

Raberto, M., S. Cincotti, S. M. Focardi, and M. Marchesi (2001). Agent-based simulation of a financial market. *Physica A: Statistical Mechanics and its Applications 299*(1-2), 319–327.

Railsback, S. F., S. L. Lytinen, and S. K. Jackson (2006). Agent-based simulation platforms: Review and development recommendations. *Simulation 82*(9), 609–623.

Rizoiu, M.-A., Y. Lee, S. Mishra, and L. Xie (2017). A tutorial on hawkes processes for events in social media. *arXiv preprint arXiv:1708.06401*.

Sela, A., D. Goldenberg, E. Shmueli, and I. Ben-Gal (2016). Scheduled seeding for latent viral marketing. In *Advances in Social Networks Analysis and Mining (ASONAM), 2016 IEEE/ACM International Conference on*, pp. 642–643. IEEE.

Serrano, E. and C. A. Iglesias (2016). Validating viral marketing strategies in twitter via agent-based social simulation. *Expert Systems with Applications 50*, 140–150.

Serrano, E., C. Á. Iglesias, and M. Garijo (2015). A novel agent-based rumor spreading model in twitter. In *Proceedings of the 24th International Conference on World Wide Web*, pp. 811–814. ACM.

Shaman, P. (1975). An approximate inverse for the covariance matrix of moving average and autoregressive processes. *The Annals of Statistics*, 532–538.

Shao, H., S. Wang, S. Li, S. Yao, Y. Zhao, T. Amin, T. Abdelzaher, and L. Kaplan (2017). Optimizing source selection in social sensing in the presence of influence graphs. In *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*, pp. 1157–1167. IEEE.

Wang, D., M. T. A. Amin, S. Li, T. F. Abdelzaher, L. M. Kaplan, S. Gu, C. Pan, H. Liu, C. C. Aggarwal, R. K. Ganti, X. Wang, P. Mohapatra, B. K. Szymanski, and H. K. Le (2014). Using humans as sensors: An estimation-theoretic perspective. *IPSN-14 Proceedings of the 13th International Symposium on Information Processing in Sensor Networks*, 35–46.

Wang, S., L. Su, S. Li, S. Hu, M. T. A. Amin, H. Wang, S. Yao, L. M. Kaplan, and T. F. Abdelzaher (2015). Scalable social sensing of interdependent phenomena. In *IPSN*.

Yao, S., M. T. Amin, L. Su, S. Hu, S. Li, S. Wang, Y. Zhao, T. Abdelzaher, L. Kaplan, C. Aggarwal, et al. (2016). Recursive ground truth estimator for social data streams. In *Proceedings of the 15th International Conference on Information Processing in Sensor Networks*, pp. 14. IEEE Press.

Yao, S., S. Hu, S. Li, Y. Zhao, L. Su, L. M. Kaplan, A. Yener, and T. F. Abdelzaher (2016). On source dependency models for reliable social sensing: Algorithms and fundamental error bounds. *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*, 467–476.

Zhang, J., L. Tong, P. Lamberson, R. Durazo-Arvizu, A. Luke, and D. Shoham (2015). Leveraging social influence to address overweight and obesity using agent-based models: the role of adolescent social networks. *Social science & medicine 125*, 203–213.

Zhao, Q., M. A. Erdogdu, H. Y. He, A. Rajaraman, and J. Leskovec (2015). Seismic: A self-exciting point process model for predicting tweet popularity. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1513–1522. ACM.