Efficient parallel simulation over large-scale social contact networks

Wu, Yulin; Cai, Wentong; Li, Zengxiang; Tan, Wen Jun; Hou, Xiangting

2019

Wu, Y., Cai, W., Li, Z., Tan, W. J., & Hou, X. (2019). Efficient parallel simulation over large-scale social contact networks. *ACM Transactions on Modeling and Computer Simulation*, 29(2), 10-. doi:10.1145/3265749

<https://hdl.handle.net/10356/143058>

<https://doi.org/10.1145/3265749>

© 2019 Association for Computing Machinery. All rights reserved. This paper was published in *ACM Transactions on Modeling and Computer Simulation* and is made available with permission of Association for Computing Machinery.

Downloaded on 26 Apr 2023 19:59:08 SGT

Efficient Parallel Simulation over Large-scale Social Contact Networks

YULIN WU, Nanyang Technological University, Singapore

WENTONG CAI, Nanyang Technological University, Singapore

ZENGXIANG LI, Institute of High Performance Computing (A*STAR), Singapore

WEN JUN TAN, Nanyang Technological University, Singapore

XIANGTING HOU, Nanyang Technological University, Singapore

Social contact network (SCN) models the daily contacts between people in real life. It consists of agents and locations. When agents visit a location at the same time, the social interactions can be established among them. Simulations over SCN have been employed to study social dynamics such as disease spread among population. Because of the scale of SCN and the execution time requirement, the simulations are usually run in parallel. However, a challenge to the parallel simulation is that the structure of SCN is naturally skewed with a few hub locations that have far more visitors than others. These hub locations can cause load imbalance and heavy communication between partitions, which therefore impact the simulation performance. This paper proposes a comprehensive solution to address this challenge. Firstly, the hub locations are decomposed into small locations, so that SCN can be divided into partitions with better balanced workloads. Secondly, the agents are decomposed to exploit data locality, so that the overall communication across partitions can be greatly reduced. Thirdly, two enhanced execution mechanisms are designed for locations and agents respectively to improve simulation parallelism. To evaluate the efficiency of the proposed solution, an epidemic simulation was developed and extensive experiments were conducted on two computer clusters using three SCN data sets with different scales. The results demonstrate that our approach can significantly improve the execution performance of the simulation.

CCS Concepts: • **Computing methodologies** → **Parallel algorithms; Simulation types and techniques;**

Additional Key Words and Phrases: social network; parallel and distributed simulation; load balancing; skewed degree distribution; graph partitioning

ACM Reference Format:

Yulin Wu, Wentong Cai, Zengxiang Li, Wen Jun Tan, and Xiangting Hou. 2018. Efficient Parallel Simulation over Large-scale Social Contact Networks. *ACM Trans. Model. Comput. Simul.* 1, 1, Article 1 (January 2018), 25 pages. <https://doi.org/0000001.0000001>

1 INTRODUCTION

A social network refers to a set of individuals and their social interactions [Borgatti et al. 2009]. Different from online social network such as Facebook, *Social Contact Network* (SCN) in this paper refers in particular to the interactions in real life [Eubank et al. 2004; Zhang et al. 2015c]. For example, when multiple persons travel to a location at the same

Authors' addresses: Yulin Wu, Nanyang Technological University, Singapore, wuyulin2005@hotmail.com; Wentong Cai, Nanyang Technological University, Singapore, aswtcai@ntu.edu.sg; Zengxiang Li, Institute of High Performance Computing (A*STAR), Singapore, liz@ihpc.a-star.edu.sg; Wen Jun Tan, Nanyang Technological University, Singapore, wtan047@e.ntu.edu.sg; Xiangting Hou, Nanyang Technological University, Singapore, houx0009@e.ntu.edu.sg.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2018 Association for Computing Machinery.

Manuscript submitted to ACM

Manuscript submitted to ACM

Fig. 1. Social contact network modelled by agent-to-location (A2L) bipartite graph.

time, there can be face-to-face interactions at that location. SCN can be represented by an agent-to-location (A2L) graph [Barrett et al. 2008], as shown in Fig. 1. The graph has a bipartite structure which consists of agents and locations. Each edge from an agent to a location represents the agent’s visit to the location during a time period.

SCN is an important component of virtual city [Ge et al. 2014] which simulates people’s daily activities. Simulations over SCN (referred to as SCN simulations) have been applied to study social dynamics such as disease spread [Eubank et al. 2004] and information diffusion [Apolloni et al. 2009]. The study usually involves two types of agents namely *Carrier* and *Receiver*, for example, the infectious and susceptible agents in epidemic simulation, or active and inactive agents in information propagation [Jin et al. 2012]. The *contact* between agents refers to the disease/information transmission from a carrier to a receiver. During the simulation, contacts are computed at each location for its visitors with overlapped visit time.

It has been observed that networks derived from real world typically have skewed degree distributions [Gonzalez et al. 2012], which means that a few nodes are connected to a large number of neighbors while most nodes are connected to a few neighbors. SCN also exhibits such kind of skewed structure. Specifically, popular locations such as metro stations and shopping malls have a huge number of visitors. On the contrary, for large numbers of home locations, each usually contains only a few members. The high-degree nodes are called *hub nodes*. The hub nodes of SCN have a strong impact on social dynamics [Bisset et al. 2011; Wang et al. 2011]. For example, the hub nodes can accelerate disease spread due to large numbers of interactions [Zhang et al. 2010]. This also implies that the simulation of hub nodes incurs a high computational cost.

Typically, a SCN is of a large scale, which can contain the population of a city or even a nation [Yeom et al. 2014]. Meanwhile, a SCN simulation may need to be executed many times for various scenarios or parameter combinations [Bisset et al. 2014, 2009]. In order to scale-up the simulation and gain insights from the simulation in a timely manner, the SCN simulation can be partitioned onto multiple processors and run in parallel [Barrett et al. 2008]. However, efficient parallel simulation of large-scale networks with skewed degree distribution has been recognized as a challenge [Fujimoto 2016; Pienta and Fujimoto 2013]. The workloads among processors are hard to be balanced because a few hub nodes have a large computation cost. Hence, the efficiency and scalability of parallel simulation are impaired.

Yeom et al. [2014] proposed an application-level method to address the above mentioned load-balancing challenge. It decomposes a hub location into small rooms before simulation and assumes that contact cannot be created between agents in different rooms. Hence, the degree distribution of the underlying network is no longer skewed after the transformation. However, this approach cannot be applied to the scenarios containing large public places where agents move around freely. For example, although the work place of a building can be divided into small offices, the same approach cannot be applied to the metro station and shopping mall due to the dynamic crowd movement. Compared

with their approach, the solution proposed in this paper allows very large locations to be simulated and meanwhile addresses the challenge of simulation performance.

Our previous work [Wu et al. 2017] improves the load balancing of SCN simulation by decomposing a hub location into split locations. Different from the approach proposed in [Yeom et al. 2014], these split locations cooperate together to enable contacts between agents in different split locations. In this way, the underlying network topology is unaltered. As an extension of our previous work, this paper proposes a comprehensive solution to improve the simulation performance. Besides the location decomposition, we further decompose agents. Since a hub location has a large number of visitors, there is a very high probability that agents and their visited hub location's splits are placed at different partitions. This will cause heavy communication overhead during the simulation. To reduce the overhead, we decompose an agent to localize the communication if it visits locations at different partitions. Furthermore, our previous work considers only static partitioning, which has limitations when handling dynamic workload. For example, in the contact computation for disease spreading, there can be a sudden surge of workload if many agents visiting a hub location are infected in a short period of time. Therefore, in this paper we further enhance the execution of locations and agents by using offloading and asynchronous stepping mechanisms to explore the simulation parallelism, so that the impact of load imbalance caused by social dynamics during the simulation can be mitigated.

SCN is mainly applied to the study of disease spreading, and therefore this paper focuses on epidemic simulation. Nevertheless, the proposed methods can be applied to other SCN simulations which exhibit similar interaction patterns between agents and hub locations. The main contributions of our work are summarized as follows:

- An analysis of the challenges in parallel simulation over large-scale SCNs with skewed topology (Sec. 3).
- A hub location decomposition method to improve the network partitioning (Sec. 4.1).
- An agent decomposition method to reduce the communication overhead (Sec. 4.2).
- Offloading and asynchronous stepping mechanisms to improve load balancing during runtime (Sec. 4.3).
- A comprehensive experimental evaluation of our proposed methods (Sec. 5).

2 BACKGROUND

2.1 Social Contact Network

SCN can be represented by a bipartite graph $G_{PL} = (P, L, E)$, where P is a set of agents and L is a set of locations. If an agent $p \in P$ visits a location $l \in L$, there is an edge $e = (p, l, t) \in E$ between them, where t is the temporal information. Since an agent can visit a location for several times in a day, $t = \{< t_a, t_d >\}$ is a collection of time intervals, where t_a is arrival time and t_d is departure time. SCN is spatially explicit and time varying, which means that an agent carries out daily activities to move from location to location, and in turn interacts with others at the same locations during the overlapped time periods [Xia et al. 2015].

For an agent p_i , we call the edge subset $E_{p_i} = \{e | e.p = p_i, e \in E\}$ as its *Activity List*. For a location l_j , we call the edge subset $E_{l_j} = \{e | e.l = l_j, e \in E\}$ as its *Visit List*. The visit list can be inferred from activity list, and it is obvious that:

$$\sum_{p \in P} |E_p| = \sum_{l \in L} |E_l| \quad (1)$$

where $|E_p|$ is the number of visit locations from agent p , and $|E_l|$ is the number of visitors to location l . That is, $|E_p|$ is the degree of agent p and $|E_l|$ is the degree of location l . It has been observed that the degrees of agents do not show significant variance, whereas the degrees of locations vary greatly [Yeom et al. 2014].

Fig. 2. Example of SCN partitioning.

2.2 Parallel SCN Simulation

To enable parallel simulation, the first step is to partition the network into multiple partitions. Suppose G_{PL} is partitioned into k parts. Specifically, agent set P is partitioned into $P_1, P_2, \dots, P_k$, and location set L is partitioned into $L_1, L_2, \dots, L_k$. Fig. 2 shows an example of graph partitioning. When an agent and its visiting location are assigned to different partitions, there is a cut edge. That is, the partitioning is edge-cut. Each partition is assigned to a logical process (LP) and all LPs will run concurrently. Specifically, LP_i is allocated with P_i and L_i . It executes the agents $\{p | p \in P_i\}$ and locations $\{l | l \in L_i\}$ in turn, as shown in Algorithm 1 (abstracted from [Barrett et al. 2008]).

Algorithm 1 Synchronous SCN Simulation Algorithm

```

1: for  $t=0$ ;  $t < \text{MaxTime}$ ;  $t = t + \Delta t$  do
2:   for each agent do // Agent processing
3:     receives contact results and updates agent status;
4:     generates visit messages for the period  $(t, t + \Delta t)$ ;
5:     sends visit messages and agent status to locations;
6:   end for
7:   barrier synchronization;
8:   for each location do //Location Processing
9:     generates arrival and departure events from visit messages;
10:    computes contacts between agents;
11:    sends contact results to agents;
12:  end for
13:  barrier synchronization;
14: end for

```

The algorithm runs in time-stepped manner. Δt is the step size (Line 1). In each step, there are two phases, corresponding to *agent processing* (Line 2-6) and *location processing* (Line 8-12). During agent processing, each agent sends visit messages to locations according to its activity list. The arrival times and departure times of the visit messages are between t and $t + \Delta t$. The visit messages can be different from step to step. During location processing, each location executes a local discrete-event simulation (DES) [Bhatele et al. 2017]. The local DES firstly converts the visit messages in its visit list into arrival and departure events, which are saved in an event queue by time order (Line 9). Then it identifies agent-to-agent interactions based on the events and computes the contact results (Line 10). The identification of the interaction can be conducted using a collection: 1) arrival event adds the agent into the collection; 2) departure event removes the agent from the collection, and meanwhile computes the contacts with all other agents in the collection (whose states are different from that of the departure agent). The actual contact computation depends on the specific application. For example, in epidemic simulation it refers to the determination of disease transmission between agents. When an agent p visits a location l in a different partition, communication across partitions along the cut edge happens.

Synchronizations (Line 7 & 13) are required at the end of each phase to ensure that all messages across partitions have arrived at the receiver sides before the next phase is executed.

In Algorithm 1, all locations in the same time-step are logically executed in parallel. An agent can visit several locations in one time-step and its state can be changed by others at any location. To ensure the correctness, the change at one location should not influence the contact results at other locations. This is guaranteed by the *lookahead*, which refers to the delay period that contact result takes effect. For example, in disease transmission, there is a period during which, the newly infected agent cannot infect others. Formally, all locations can be computed in parallel only if $lookahead \geq \Delta t$.

2.3 Application - Epidemic Simulation

SCN simulations are widely used to study disease transmission among large population [Barrett et al. 2008; Grefenstette et al. 2013]. This paper uses an agent-based model of disease transmission as a case study. It consists of two related parts: *Within-host Progression Model* and *Between-host Propagation Model*. The former describes the disease progression within an infected person. The latter describes the transmission probability between two contacting persons.

Within-host Progression Model. SEIR is one of the most classical models [Kuznetsov and Piccardi 1994]. Each person can be in one of the four states at any time: *susceptible*, *exposed*, *infectious* and *recovery*. *Susceptible* denotes the state susceptible to the disease. *Exposed* is the state after being infected but without the ability to infect other *susceptible* individuals. *Infectious* stands for the state of being capable of spreading the diseases to other *susceptible* individuals. *Recovery* denotes the state of being recovered from the disease. The (minimum) duration t_e between *exposed* state and *infectious* state can be used as the simulation lookahead introduced in Algorithm 1, which is usually considered to be longer than a day. Accordingly, in the synchronous epidemic simulation, the step size Δt is typically set as one day.

Between-host Propagation Model. If two persons stay in the same location l with overlapped visit time, then a contact is established. Assuming person u 's health state is *infectious* and she/he contacts with *susceptible* person v , the probability that person v gets infected is calculated by [Bisset et al. 2009]:

$$p_r(u, l, v) = \frac{1}{c_e} \times (1 - (1 - r)^{t(u, l, v)}), \quad (2)$$

where r is the probability of disease transmission for contact per unit time. $t(u, l, v)$ is the contact duration between u and v at location l . Specifically, it is the overlapped time period between their arrival and departure events. Similar to [Zhang et al. 2015b], c_e is a corrective coefficient to adjust the disease transmission probability. During simulation, if a generated random number is smaller than $p_r(u, l, v)$, the susceptible agent v will be infected.

3 CHALLENGES

Efficient parallel simulation requires the workloads among partitions to be balanced. Otherwise, the overloaded LPs will make other LPs idle during synchronization. Meanwhile, the communication overhead amongst LPs should be minimized. The partitioning problem can be formulated as:

$$\begin{aligned} \min \quad & |\bar{E}| = |\{ \langle p, l, t \rangle \}|, \quad \forall p \in P_i \wedge \forall l \in L_j \wedge i \neq j \\ \text{s.t.} \quad & \begin{cases} W_{P_i} = \sum_{p_j \in P_i} |E_{p_j}| \leq (1 + \epsilon) \frac{\sum_{p \in P} |E_p|}{k} \\ W_{L_i} = \sum_{l_j \in L_i} f(|E_{l_j}|) \leq (1 + \epsilon) \frac{\sum_{l \in L} f(|E_l|)}{k} \end{cases} \end{aligned} \quad (3)$$

where $|\bar{E}|$ is the total number of cut edges. Intuitively, minimizing $|\bar{E}|$ will minimize the communication overhead amongst LPs. W_{p_i} represents the overall workload of agents in partition i and the workload of an agent p is estimated using its degree $|E_p|$. To approximately achieve load balancing, W_{p_i} should not be larger than the average workload multiplied by $(1 + \epsilon)$, where ϵ is a tolerance factor. W_{L_i} represents the overall workload of locations in partition i . Different from agent, the workload of location can have a non-linear relation to its degree [Yeom et al. 2014]. So, a function f of location's degree $|E_l|$ is used to estimate the workload of location l . Similarly, to achieve load balancing, W_{L_i} should not be larger than the average workload multiplied by $(1 + \epsilon)$.

Existing graph partitioning tools can be used to achieve above objective. For example, METIS [Abou-Rjeili and Karypis 2006] uses multi-constraint k -way partitioning algorithm to ensure load balancing of different computing phases. Hence, both the agent and location processing can be considered simultaneously during partitioning. It also tries to minimize the number of cut edges (referred to as *minimum cut*). However, the hub locations in SCN pose challenges to graph partitioning due to both their computation and communication demands.

The first problem is **Load Balancing**. Fig. 3a shows an example of location workload distribution per degree, which is based on an epidemic simulation described in Section 5. The workload is measured by average execution time per time step, which shows a non-linear relation to the degree. For the locations with high degrees, their workloads scatter across a very wide range. This is because the workload of a location is not only determined by its degree, but also the social dynamics of the visitors (e.g., staying time and disease status). Consequently, the piecewise linear regression method [Yeom et al. 2014], which is usually used in practice due to its simplicity, cannot estimate the workloads of hub locations accurately. Hence, it may cause imbalanced partitioning result when METIS is used.

Fig. 3b plots the execution time of location processing in different partitions during the simulation (only 4 partitions are shown for simplicity). As shown in the figure, the workload of each partition changes with simulation time. Specifically, the workload is much higher during 20th to 40th simulation day. The main reason is that the number of infected agents and the risk of disease transmission are much higher during this simulation time period (see Fig. 11). In addition, compared with other partitions, partition 16 has much higher workload which also changes more dramatically. This can be explained by the fact that the partition has some hub locations whose workloads are higher and vary more greatly than other locations. The static graph partitioning cannot capture the dynamism well. In fact, the location workloads among partitions should be balanced for each time-step:

$$W_{L_i}(t) = \sum_{l_j \in L_i} f(|E_{l_j}|, t) \leq (1 + \epsilon) \frac{\sum_{l \in L} f(|E_l|, t)}{k}, \quad 0 \leq t < MaxTime \quad (4)$$

where $W_{L_i}(t)$ is the total location workload of partition i at time-step t , and $f(|E_l|, t)$ is the workload of location l at time-step t .

The second problem is **Communication**. The irregular network structure causes large communication overhead. Each cut edge typically requires at least one inter-partition communication at each simulation day, i.e., sending the visit message. Although graph partitioning tools like METIS try to obtain minimum cut to minimize the communication overhead, the number of cut edges is still large when they are applied to partition SCNs. That is because the hub locations in a SCN are typically connected to a large fraction of population and the network diameter is rather small in networks with skewed degree distribution [Pržulj et al. 2004]. Gonzalez et al. [2012] demonstrated that hub nodes show a star-like motif with concentrated edges. Hence, edges of a hub location are prone to being cut. Our previous work [Wu et al. 2017] showed that about half of six million edges in an open data set [Bisset et al. 2006] are cut with only 36

Fig. 3. Workloads of locations and partitions.

Fig. 4. Overview of the proposed solution.

partitions. Furthermore, even after hub locations are split, as we will illustrate in Section 4, the number of cut edges does not show significant change in most cases.

The third problem is **Scalability**. Existing work has demonstrated that the hub locations in a SCN limit the system scalability [Yeom et al. 2014]. Since a location cannot be executed in parallel, the largest hub location becomes the bottleneck of scalability. More importantly, the average workload among partitions will decrease with the increase of partitions, so that the impact of hub locations on load balancing may become more significant.

4 PROPOSED SOLUTION

We propose a comprehensive solution to address the performance problems caused by hub locations. The solution includes three parts as shown in Fig. 4. Firstly, hub locations are decomposed into smaller locations to enhance partitioning in terms of load balancing. Secondly, agents are decomposed to reduce communication due to the cut edges and partitions are updated accordingly. Finally, the updated partitions are executed using the enhanced execution mechanisms to improve simulation parallelism.

4.1 Hub Location Decomposition

Our method of hub location decomposition is at system level, which means that the resulted split locations maintain connections so that the contacts between visitors remain the same as those in the original hub location. The decomposition is conducted before graph partitioning and involves the following steps:

Fig. 5. Hub location decomposition.

- (1) Hub locations are identified. A threshold of node degree D_T is used to select locations to be decomposed.
- (2) For each selected location l , its neighboring agents are split into $s = \min(\lceil \frac{|E_l|}{D_T} \rceil, S_T)$ groups at random, where $|E_l|$ is the degree of l , and S_T is a threshold for the allowed maximum number of splits. All groups have approximately equal number of agents. After location decomposition, the maximum node degree of the network is $\frac{|E_l|}{s}$.
- (3) For each agent group, a new split location is created. The agents in the group are rewired from the original hub location to the new split location.
- (4) For each split location, the workload can be approximated by $\frac{f(|E_l|)}{s}$, where $f(|E_l|)$ is the workload of the original hub location.
- (5) The original hub locations are then removed from the network and the new network is partitioned using METIS. In order to evenly distribute the split locations into all partitions, three-phase-constraint partitioning pattern is used in METIS. The three phases correspond to the agent processing, non-split location processing and split location processing in each simulation time-step.

In order to guarantee the correctness, the contact relationship within the original location must be maintained after it is decomposed into multiple split locations. For each pair of visitors, if one is carrier and the other is receiver, then there is a possibility for the contact to be established. To maintain the original contact relationship, a solution is to make each split location keep only a proportion of receivers but all the carriers.

Fig. 5 shows the concept of hub location decomposition and the data exchange mechanism during simulation. For example, in Fig. 5a the degree of original hub location a is eight. After splitting a into b and c (Fig. 5b), the degrees of both splits are four. Suppose agents 3 and 6 are carriers, their visit messages need to be exchanged between b and c . For example, message from agent 6 will be firstly delivered to split b , which will process the message locally for receivers 1, 4 and 7. At the same time the message is forwarded to split c by b , and then c will process the message locally for receivers 2, 5 and 8. In this way, the workload at original hub location a is divided.

The approach has two advantages. First, the existing partitioning tools/methods can be utilized to partition the resulting network with some pre-determined constraints. Second, the number of splits can be limited by the threshold D_T and S_T , so that the additional communication overhead among splits for forwarding carriers' visit messages can be controlled.

The decomposition enables parallel execution of a hub location because its splits can be assigned to different partitions. As mentioned above, METIS is used for partitioning, but we increase the parallelism by the following ways:

Table 1. Operation abstraction in SCN simulation

Operation	Node type	Description
Receive	Agent	Receives contact results and updates status
	Split and non-split location	Receives visit messages and converts them into arrival/departure events
Compute	Agent	Generates visit messages
	Split and non-split location	Computes the contact result according to arrival/departure events
Send	Agent	Sends visit messages to locations
	Split and non-split location	Sends contact results to agents
Forward	Split location	Forwards visit messages from carriers to other splits of the same location

- (1) Hub locations are decomposed into parts as many as possible by adjusting D_T and S_T . Intuitively, the more splits, the more likely they are assigned to different partitions.
- (2) The input graph to METIS does not contain the links between split locations of the same hub location, so that METIS treats them as separated nodes. The links are rebuilt after the partitioning.
- (3) An additional load-balancing constraint for split location processing phase is added when using METIS, so that split locations are evenly assigned to partitions.

The decomposition can improve load balancing because each location now has a lighter workload and thus a smaller load variation (see Fig. 3b). However, additional communication overhead is introduced, because the visit messages of carriers need to be forwarded from one split to all other splits of the same location. The communication overhead is linearly correlated to the number of visit messages of carriers per split. Moreover, the message forwarding introduces an extra phase which also requires synchronization. To ensure correctness, this phase should be placed between agent and location processing phases.

In order to reduce the overhead, we can overlap the message forwarding of splits with the processing of non-split locations. This requires the separation of message handling from the location processing. We abstract the processing of each LP by *Receive*, *Compute* and *Send* operations, corresponding to the Algorithm 1. We also introduce a new operation *Forward* to represent the message forwarding among split locations of the same hub location. Tab. 1 summarizes all operations.

Algorithm 2 shows the modified procedure to handle split locations. At the first step of location processing phase, *Receive* operation will be executed for splits (Line 7). If there are carriers, their visit messages will be forwarded to other splits of the same location by *Forward*. After that, a *FORWARD_COMPLETE* message is broadcasted at Line 9 to indicate the completion of forwarding phase of the current LP. Then, the non-split locations will be processed (Line 10-12). After the processing of non-split locations is done, a synchronization is required to ensure the completion of the forwarding phase (Line 13). This synchronization is not implemented using a global barrier. As long as *FORWARD_COMPLETE* messages from all other LPs have been received, the synchronization is done. Finally, the *Compute* and *Send* operations of the split locations are executed (Line 14-16).

An LP does not need to wait for the completion of the forwarding phase. When the processing of non-split locations is done, an LP should have received *FORWARD_COMPLETION* messages from all other LPs in most cases, so that the synchronization of the forwarding phase will not generate (much) idle time.

Algorithm 2 Modified Synchronous Algorithm to Handle Split Location

```

1: for  $t=0; t < \text{MaxTime}; t = t + \Delta t$  do
2:   for each agent do //Agent processing
3:     Call Receive()  $\rightarrow$  Compute()  $\rightarrow$  Send();
4:   end for
5:   barrier synchronization;
6:   for each split location do //Split location forwarding
7:     Call Receive()  $\rightarrow$  Forward();
8:   end for
9:   broadcast FORWARD_COMPLETE message;
10:  for each non-split location do //Non-split location processing
11:    Call Receive()  $\rightarrow$  Compute()  $\rightarrow$  Send();
12:  end for
13:  wait to receive FORWARD_COMPLETE message from all other LPs;
14:  for each split location do //Split location processing
15:    Call Compute()  $\rightarrow$  Send();
16:  end for
17:  barrier synchronization;
18: end for

```

Fig. 6. Messages to location dominated communication pattern.

4.2 Agent Decomposition

The cut edge resulting from graph partitioning incurs inter-partition communication. As discussed in Section 3, edges of a hub location are prone to being cut. Even if hub locations are decomposed, there are still many cut edges. This is because the split location's degree ($\frac{|E_H|}{s}$) is still much higher than the average degree. As a result, the star-like motif is not removed. We observed that the two-direction communication along the edge is not symmetric and the communication is dominated by the messages from agents to locations as shown in Fig. 6. Specifically, in each simulation day all agents need to carry out their daily activities by sending visit messages to locations (Fig. 6a). In contrast, a location only sends contact results to a few newly infected agents (Fig. 6b). In view of this, if we ensure that the messages to locations are generated locally, hub locations will not suffer from large communication overhead due to edge-cut.

Given that an agent is scheduled by its activity list, we can divide the activity list to achieve data locality when its edges are cut. This is accomplished by agent decomposition. Fig. 7 shows an example with respect to Fig. 2. When the edge from agent 2 to location c is cut, a node is created in partition 2 to represent agent 2. Now there are two nodes representing the same agent, but they are not equal because each maintains only local activity list in the current

Fig. 7. Converting cut edges into split agents (w.r.t. Fig. 2).

partition. They are referred to as splits of the agent. More generally, an agent can be decomposed into s splits if its visiting locations are placed in s partitions.

Since agents (activity lists) are decomposed, there is no visit message which needs to pass across partitions. Therefore, the first synchronization at Line 5 of Algorithm 2 can be removed. However, a new communication message is introduced for state synchronization among splits of the same agent. Nevertheless, this is required only when the agent's state is changed and can be achieved if the location sends contact result directly to all the splits of the agent.

To estimate the reduced communication volume for epidemic simulation, we suppose the number of cut edges after graph partitioning is $|\bar{E}|$, the number of simulation steps is T days, and the disease attack rate (i.e., the ratio of total infected people to total population) is α . We further assume that each agent can be infected only once because usually T is considered to span a short period of time [Barrett et al. 2008]. Then, the original communication volume is $|\bar{E}| \times T$. Using agent decomposition, the communication volume becomes $|\bar{E}| \times \alpha$. Generally, $T \gg 1 \gg \alpha$, the communication overhead could be reduced dramatically.

Agent decomposition is made after graph partitioning, and hence it does not influence the hub location decomposition. Agent decomposition is different from agent replication. The latter has been used in agent-based simulation platforms such as Repast HPC [Collier and North 2013]. For agent replication, an exact copy of an agent is created at a neighboring LP so that it can be accessed locally. The replica and the original agent have exactly the same data. In contrast, the splits of an agent in our case are different from each other because they have different data (i.e., local activity list) and thus they can have different behaviors.

4.3 Execution Enhancement

Using hub location decomposition, the workload of a hub location can be distributed to multiple LPs to obtain better load balancing. However, it is a static method and cannot handle load imbalance caused by social dynamics during the simulation. A possible solution is to migrate simulated entities, such as agents and locations in SCN, to dynamically balance the workload. However, this is not efficient for SCN simulation because 1) the execution of an agent or non-hub location, both of which make up about 90% of SCN, takes very little time (typically dozens of microseconds); and 2) the migration of a large number of small entities may incur large overhead. Therefore, instead of migrating agents and locations, we balance the load during runtime using offloading and asynchronous stepping techniques as described below.

For locations, we employ an offloading technique. When an LP is overloaded, we move the *Compute* operation of some split locations into an underloaded LP by transferring their visit lists. The visitors and split locations are still managed by the source LP. The target LP is only responsible for computing contacts (using the visit lists) and returning

Fig. 8. Offloading scheme using pairwise matching.

the contact results back to the source LP. So, graph re-partitioning and migration of entities are not necessary. We only offload split location because it needs much more execution time than non-split location and agent. The approach consists of the following steps.

- (1) Monitor the workload. At each LP, the total execution time of each time-step is recorded. In addition, the execution time of each split location is also recorded because it will be the basic unit of the offload operation.
- (2) Broadcast the workload information. Each LP sends its total execution time to others in each time-step. Hence, all LPs know the global load situation.
- (3) Decide an offloading scheme. Since the global load situation is known by all LPs, a distributed algorithm can be executed in each LP. If the current LP is overloaded, the output of the algorithm is an offloading scheme including the target LP and split locations to be offloaded.
- (4) Execute the offloading scheme. In the next iteration, the selected split locations are offloaded from the source LP. Specifically, the visit lists of split locations in the scheme are sent to the target LP.
- (5) Execute the offloaded split locations. In the target LP, once an offloaded split location (i.e., visit list) is received, it executes the split location immediately and returns the contact results to the source LP together with its execution time.
- (6) Handle the execution result of offloaded split locations. The source LP waits for the contact results of the offloaded split locations before sending the synchronization message for the current step. Once the contact results are received, they are used to update agents. The execution time of an offloaded split location will be counted as the source LP's workload.

The generation of offloading scheme in the third step is important because it determines how the target LP is selected and how much workload should be offloaded. The procedure is executed on each LP. To ensure a consistent solution, we use a pairwise matching method as shown in Fig. 8. In this method, we firstly sort all LPs by their workloads in descending order, then recognize the overloaded and underloaded LPs according to the average workload; and finally match overloaded LP with underloaded LP by their respective positions in the sorted list. For example, the first LP is matched with the last LP, and the second LP is matched with the second last LP and so on.

Algorithm 3 shows the detailed generation of offloading scheme. Firstly, the average load among LPs is calculated according to the global load information (Line 1). If the current LP is underloaded, the algorithm returns without offloading scheme (Line 2-4). Secondly, LPs are sorted by their loads in descending order and are put in *SortedLPList* (Line 5). Then, the source LP p is matched with the target LP q (Line 6-8). The matching is determined by their positions

in *SortedLPList*. If the target LP is overloaded, then the algorithm returns without offloading scheme (Line 9-11). Thirdly, to determine split locations to be offloaded, the load level that can be offloaded (T_{off}) is calculated (Line 12) and the local split locations are sorted in the queue S by their execution times (Line 13). Then, the split locations to be offloaded are selected (Line 14-19). The *MinExecutionTime* in Line 16 is to ensure that the split locations with small loads are excluded because of the overhead of offloading. It is set to 500 milliseconds in our experiments. Finally, the offloading scheme with the target LP q and the set of split locations for offloading O is returned.

Algorithm 3 Generation of Offloading Scheme

Input: k : number of partitions; p : current LP; $T_1, \dots, T_k$: execution times of LPs; S : list of split locations;

Output: q : target LP, initialized to NIL; O : set of split locations to be offloaded, initialized to $\emptyset$;

```

1:  $T_{avg} = (T_1 + \dots + T_k)/k$ ;
2: if  $T_{avg} > T_p$  then
3:   return; //Source LP is underloaded and no split location will be offloaded
4: end if
5:  $SortedLPList$  = list of LPs sorted by execution time in descending order;
6:  $i$  = index of  $p$  in  $SortedLPList$ ; //  $1 \leq i \leq k$ 
7:  $j = k + 1 - i$ ;
8:  $q = SortedLPList[j]$  //Target LP for offloading
9: if  $T_{avg} < T_q$  then
10:   $q = NIL$ ; return; //Target LP is overloaded and no split location will be offloaded
11: end if
12:  $T_{off} = \min(T_p - T_{avg}, T_{avg} - T_q)$ ;
13: Sort  $S$  by execution time in descending order;
14: while  $T_{off} > 0$  do
15:   $l = S.pop()$ ;
16:  if  $l.ExecutionTime < MinExecutionTime$  then break;
17:   $T_{off} = T_{off} - l.ExecutionTime$ ;
18:   $O = O \cup l$ ;
19: end while
20: return;

```

The algorithm is executed in each iteration with small overhead. The computing complexity mainly comes from the sorting operations of LPs (Line 5) and split locations (Line 13), which is $O(k \log k + |S| \log |S|)$ where both k and $|S|$ are small.

Next, we enhance the execution of agents by asynchronous stepping (referred to as *AsyncStep*) which reduces the idle time caused by load imbalance during synchronization. The underlying idea is that a simulation entity can be updated in advance before synchronization as long as its dependent data are available [Zhang et al. 2015a; Zou et al. 2011]. Fig. 9 shows the basic concept of *AsyncStep*. With agent decomposition there is no visit message that needs to be sent to other partitions. So, synchronization is no longer required after agent processing. After location processing, an LP will send a *LOCATION_COMPLETE* message to all other LPs. Then, it will immediately execute agent processing of next time-step before performing location synchronization for the current time-step. An LP will progress to the next time-step as long as it receives *LOCATION_COMPLETE* messages from all other LPs. In this way, as shown in Fig. 9b, agent processing of time-step $t + 1$ of one LP can overlap with location processing of time-step t of another LP and the idle time caused by load imbalance during synchronization can hence be reduced.

Fig. 9. Reducing idle time by asynchronous stepping.

It is possible to execute agent processing in advance because the within-host progression model determines disease progression of an agent using only the data local to the agent. Since agent (or rather agent's split) and its visiting location are in the same partition after agent decomposition, visit message can also be sent to location in advance. However, an agent's state can be changed by the contact calculation of a location residing on a different partition, and the updated state is only (correctly) propagated to the split agent after location synchronization. To ensure the correctness of simulation, we force the location to access agent's state directly during contact calculation so that the updated agent's state is utilized. Again, this is possible because agent (or split agent) and its visiting location are in the same partition.

AsyncStep only absorbs idle time during synchronization, so the upper bound of performance improvement is $T_{sim} - \max(T_i)$. Here T_{sim} is the total simulation time without using *AsyncStep*, and T_i is the actual execution time (i.e., excluding idle time) of LP_i . If an overloaded LP has the longest execution time in most steps, then *AsyncStep* cannot bring much benefit because $T_{sim} - \max(T_i)$ is small. Therefore, *AsyncStep* should be used in combination with offloading technique to achieve greater performance improvement.

The final simulation algorithm with location decomposition, agent decomposition and execution enhancement is shown in Algorithm 4. The offloading scheme is generated at the end of each iteration (Line 23) according to the global load situation. In the next iteration, the scheme will be executed (Line 13) to offload the selected split locations to the target LP. When the target LP receives the offloaded split locations, it will handle them in the background (not shown in the algorithm) by periodically checking the received messages. The source LP will wait to receive and handle the contact results returned from the target LP (Line 17) before broadcasting *LOCATION_COMPLETE* message to all other LPs.

Agent processing for the next time-step (Lines 19-21) is immediately executed after broadcasting *LOCATION_COMPLETE* message (Line 18) and before location synchronization. The location synchronization is considered achieved once an LP receives the *LOCATION_COMPLETE* messages from all other LPs. It should be noted that agent's state is updated by location directly and the update message to a remote split agent is also handled in the background when periodically checking the received messages, so *Receive* operation is no longer needed in agent processing. It should be also noted that barrier synchronization is no longer required during the simulation.

Algorithm 4 Simulation Algorithm with Enhanced Execution

```

1: for each agent do //Agent processing for time-step 0
2:   Call Compute()  $\rightarrow$  Send();
3: end for
4: for  $t=0$ ;  $t < \text{MaxTime}$ ;  $t = t + \Delta t$  do
5:   for each split location do //Split location forwarding
6:     Call Receive()  $\rightarrow$  Forward();
7:   end for
8:   broadcast FORWARD_COMPLETE message;
9:   for each non-split location do //Non-split location processing
10:    Call Receive()  $\rightarrow$  Compute()  $\rightarrow$  Send();
11:   end for
12:   wait to receive FORWARD_COMPLETE message from all other LPs;
13:   send offloaded split locations to target LP; //Execute offloading scheme
14:   for each split location do //Split location processing
15:     Call Compute()  $\rightarrow$  Send();
16:   end for
17:   receive and handle the results of offloaded split locations;
18:   broadcast LOCATION_COMPLETE message with information on execution time for time-step  $t$ ;
19:   for each agent do //Agent processing for time-step  $t+1$ 
20:     Call Compute()  $\rightarrow$  Send();
21:   end for
22:   wait to receive LOCATION_COMPLETE message from all other LPs;
23:   generate offloading scheme; //Algorithm 3
24: end for

```

5 EXPERIMENT**5.1 SCN Data Sets**

An open data set [Bisset et al. 2006] published by Network Dynamics and Simulation Science Laboratory (NDSSL) was employed in the experiments. The version in use is *Release 1*, which is a synthetic population of the city of Portland. The data set has 1,615,860 people and 181,295 locations. The location types include home, work place, school, shopping mall and social place. The demographic attributes of the synthetic population are consistent with the census data and the locations are generated according to the transportation network in the city. The activity patterns assigned to each synthetic person are taken from a survey of the residents and involve only a normative day. A few shopping malls and social places have thousands of visitors. This data set is firstly reported in the work [Eubank et al. 2004] and later used by several studies related to epidemic simulation [Barrett et al. 2005; Eubank 2005; Guo 2007].

Because the original data set is a small city (referred to as *S* data set), we enlarged it by four and eight times respectively to obtain the middle and large data sets (referred to as *M* and *L* respectively). To enlarge the data set, homes and work places are added when the city population increases; whereas the large public places such as shopping malls are shared by both old and new population. Taking data set *M* for example, it is created by the following steps: 1) Create four copies of the original data set; 2) For shopping malls and social places with degree larger than 2000 in the last three copies, re-direct people visiting these locations to their counterparts in the first copy; and 3) For the locations with degree between 1000 and 2000 in the last three copies, randomize the visits amongst all locations. After performing the above steps, the *M* data set has 6,463,440 people and the largest location has about 24,770 visits. As for the *L* data set, it has 12,926,880 people and the largest location has about 47,884 visits.

Table 2. Information of Data Sets

Property	Value
People	1,615,860(<i>S</i>); 6,463,440(<i>M</i>); 12,926,880(<i>L</i>)
Locations	181,295(<i>S</i>); 725,180 (<i>M</i>); 1,450,360(<i>L</i>)
Edges	6,108,721(<i>S</i>); 24,484,994(<i>M</i>); 49,008,965(<i>L</i>)
Visits	8,922,359(<i>S</i>); 35,689,436(<i>M</i>); 71,378,872(<i>L</i>)
Visits of The Largest Location	7,436 (<i>S</i>); 24,770(<i>M</i>); 47,884(<i>L</i>);
Visit Types	Home, Work, Shop, Social, etc.
Period of Activity	1 day

Fig. 10. Location's degree distribution of each data set (in log-log scale).

Tab. 2 lists the details of the three data sets. The *Visits* is larger than the *Edges*, because one person may visit one location more than once. The *Visit Type* is a property of each visit and implies the type of the visited location. The *Period of Activity* each data set covers is one day. So, the visit activities will be repeated by agents in each simulation day. Fig. 10 shows the location's degree distributions of the data sets. Number of visits per location is used to plot the graphs instead of the number of edges connected to the location, since it is better correlated with the workload [Yeom et al. 2014]. We can observe that the data sets *M* and *L* maintain the skewed degree distribution as the original data set *S*.

5.2 Experimental Setup

We firstly used a small cluster to perform basic performance evaluation. The cluster has six machines, each of which has two Intel Xeon E5-2620 v3 CPUs. The total number of CPU cores is 72. The machines are installed with CentOS 6.6 and are connected by 10G Ethernet. The SCN was partitioned by METIS 5.1.0 and the parallel epidemic simulation over SCN was implemented using MPICH2 3.2. Specifically, each partition is mapped to an LP, which is executed by an MPI process on one CPU core. To reduce the overall communication overhead, small messages are aggregated in buckets before sending via MPI. When a bucket is full or simulation synchronization is called, all messages in the bucket are flushed immediately.

To examine the effect of each method in our solution, we implemented four programs which are named *Original*, *Split_Location*, *Split_Location_Agent* and *Split_Enhance* (see Tab.3). The *Original* represents the naive method as presented in Algorithm 1, and other three variations use the proposed methods in this paper. *Split_Enhance* represents the overall solution including all the proposed methods.

The default experimental parameters are shown in Tab. 4. 60 cores are used to run LPs, and each machine is configured to execute 10 LPs. Two cores in each machine are reserved for MPI communication and data collection. Specifically, MPI

Table 3. Variations of experimental programs

Experimental program	Location decomposition	Agent decomposition	Execution enhancement
Original			
Split_Location	√		
Split_Location_Agent	√	√	
Split_Enhance	√	√	√

Table 4. Default experiment settings

Parameter	Value
Number of LPs (k)	60
Split Threshold (D_T)	500(S), 1500(M), 2000(L)
Maximum splits per location (S_T)	15
Simulation Steps ($MaxTime$)	120(Days)
Step Size (Δt)	1(Day)
Initial Infected People	100
Disease Incubation Period (t_e)	2(Days)
Disease Infectious Period	4(Days)
Transmission Probability (r)	0.3
Transmission Corrective Coefficient (c_e)	55

messages are received by a dedicated thread but the message handling is interleaved and executed by the main thread running the simulation algorithm (e.g., Algorithm 1 or 4). All experiments of performance evaluation were executed for 10 times and the mean value is used for analysis. The maximum coefficient of standard deviation is less than 5%. The initial infected people are randomly selected from the whole population. The recovered agents are removed from the simulation, that is, each agent can be infected only once. The transmission corrective coefficient c_e is adjusted so that the attack rate of the epidemic model (i.e., the ratio of total infected people to total population) is about 40%, similar to the values adopted in the related work [Barrett et al. 2008; Grefenstette et al. 2013].

5.3 Experimental Results and Analysis

5.3.1 Correctness of the Solution. To verify that our solution generates similar results to that of the original method, we compare the number of the infected people produced by *Split_Enhance* and *Original* using data set S. Given the randomness in the epidemic model, two simulation instances should not produce exactly the same results. We run each method for 10 times. Fig. 11 shows the average number of newly infected people among 10 runs in each simulation step. As it can be seen from the figure, two curves are almost overlapped. Furthermore, the average attack rate generated by using either *Original* or *Split_Enhanced* method is about 40% and the difference is less than 0.2%.

5.3.2 Basic Performance Comparison. Fig. 12a shows the execution times of each method using the three data sets. Fig. 12b shows the corresponding performance improvements over the *Original* method. We can see that location decomposition reduces the execution time by 14~18%, agent decomposition reduces the execution time by 13~18%, and the execution enhancement further reduces the execution time by 9~11%. The overall performance improvement is about 37~46%.

Fig. 11. Number of newly infected people in each simulation step (S data set).

Fig. 12. Simulation performance on a small cluster using default parameter setting.

To investigate the improvement on load balancing, we record the execution time of each LP at each simulation step, and then calculate the *accumulated load imbalance level* by:

$$W_{imb} = \sum_t \frac{W_{max}(t) - W_{avg}(t)}{W_{avg}(t)}, \quad (5)$$

where $W_{max}(t)$ is the maximum execution time among LPs at time-step t , and $W_{avg}(t)$ is the average execution time among LPs at time-step t . Fig. 13a shows the imbalance levels of each method on the three data sets. We can see that the hub location decomposition (i.e., *Split_Location*) can balance the loads better than the *Original* and with the execution enhancement (i.e., *Split_Enhance*) the imbalance level can be further reduced. As shown in the figure, the total improvement of load balancing is about 40~60%. Since hub locations influence load balancing, the imbalance level increases with the size of data set.

Fig. 13b shows the average number of received messages in each time-step. We can observe that the agent decomposition (i.e., *Split_Location_Agent*) greatly reduces the number of messages. The *Original* method needs to send a visit message for every cut edge at each time-step; whereas with agent decomposition, at most one message needs to be sent to all splits of an agent during the entire simulation. This is because only when an agent is infected, a message needs to be sent to all its splits and each agent is infected only once in our experiments. Fig. 12b shows that agent decomposition reduces 13~18% of execution time due to the reduced communication.

5.3.3 Impact of Parameters. The disease attack rate influences the workloads of locations. When the rate is high, the workload is low in our experiments. This is because with higher attach rate the disease will be transmitted between two

Fig. 13. Improvement of load balancing and communication.

Fig. 14. Impact of parameters on performance (Data set S).

interacting agents in less number of iterations (i.e., less number of contacts). Another reason is that the recovered agents are removed from simulation, and high attack rate means agents will be removed from the simulation faster. In this experiment, we vary the corrective coefficient c_e to obtain a wide range of attack rate and investigate the performance improvements of *Split_Enhance* over the *Original* method.

Fig. 14a shows the result of data set S (Data set M and L show similar results with S, so we do not repeat the descriptions here). When c_e is 25, the attack rate is about 68%, and there is about 40% improvement of execution time. When c_e is 200, the attack rate is about 5%, and we obtain the highest performance improvement (about 52%). It can be observed from the figure that the higher the workload of locations (i.e., the lower the attack rate), the more performance improvement that can be achieved by our solution. Although the attack rate also influences the amount of messages, the total number of messages is small due to agent decomposition and hence the impact on performance is small. In other words, the increase of performance improvement is mainly due to better load balancing.

We also investigate the impact of the number of split locations on performance by varying the splitting threshold D_T . Fig. 14b shows the performance improvements with different D_T settings. The thresholds are chosen unevenly due to the skewed degree distribution. Lower threshold results in more splits. As shown in Fig. 14b, the highest performance improvement is achieved when the threshold is 500. When there are more splits, the overhead of communication among splits has a larger impact on performance; whereas when there are less splits, the load imbalance due to hub location has a larger impact on performance. Since the maximum degree of location in data set S is about 7,000, no location is split when the threshold is 8000. It should be noted that even though no location is split, there is still

Table 5. Performance improvements of data sets synthesized using BA algorithm

New dataset	Degree threshold	Maximum degree	Performance improvement
1	1000	13411	33.6%
2	1500	18304	35.7%
3	2000	27158	42.5%
4	2500	41927	49.3%
5	3000	62250	61.8%

performance improvement due to agent decomposition and the employment of enhanced execution mechanisms. The result demonstrates that we should not excessively split hub locations.

5.3.4 Synthetic networks with BA algorithm. To further test the proposed approach on different range of workload distributions, BA algorithm [Barabási et al. 2000], which is frequently used to generate synthetic network with skewed degree distribution, is adopted to generate additional five data sets as shown in Tab. 5. To create a new data set, similar to data set M, four copies of the original data set are created and people who visit locations with degree larger than 2000 in the last three copies of the data sets are identified. Then, they are re-connected to the locations with degree larger than a predefined threshold in the first data set using the BA algorithm. In this way, the newly generated data sets are not just linearly scaled from the original data set. With the increasing threshold, the maximum degree of location increases. This implies that the load imbalance among partitions will become more severe if traditional partitioning approaches are used. Since our proposed solution can address the workload imbalance problem, its performance improvement over the *Original* method increases with the increasing threshold as shown in Tab. 5.

5.3.5 Scalability. A cluster from Singapore NSCC was used for scalability experiments. It has 10 machines connected by InfiniBand EDR and each machine has two E5-2690 v3 CPUs. The total number of CPU cores is $10 \times 2 \times 12 = 240$. 1, 5, 10, 15 and 20 LPs were assigned to each machine in our experiments and thus the total number of partitions were 10, 50, 100, 150 and 200 respectively. Other experimental parameters were set the same as those shown in Tab. 4.

Fig. 15 shows the ratio of cut edges to the total edges by the *Original* and *Split_Enhance* methods. It should be noted that only location decomposition influences the number of cut edges. Intuitively, more partitions or smaller SCN will lead to a larger ratio of cut edges. From the figure, it can be observed that the number of cut edges increases with the number of partitions and larger data set produces less number of cut edges with the same number of partitions. When the number of LPs are less than or equal to 100, the *Split_Enhance* method generates less number of cut edges than the *Original* method. When the number of LPs are larger than 100, the *Split_Enhance* method produces a few more cut edges. In general, the number of cut edges depends on the partitioning algorithm used in METIS as well as data set itself.

Fig. 16 compares the scaling performance of the *Original* and *Split_Enhance* methods. The execution time of both methods decreases with increasing number of LPs. The increasing trend is more obvious with larger data sets. We can also see that the *Split_Enhance* method always has less execution time than the *Original* method. Data sets S, M and L obtain 30~54%, 19~56% and 7~68% improvement respectively. The performance improvement increases with the number of LPs and the increment becomes larger with bigger data sets. The *Split_Enhance* method improves the *Original* method in terms of load balancing and reduction of communication overhead. Load balancing depends on several factors, including the number of LPs, the total workload and the degrees of hub locations. With small number

Fig. 15. Ratio of cut edges to total edges.

Fig. 16. Scaling performances on NSCC cluster.

of LPs, more balanced partitions can be generated with large data set than with small ones. This explains why the improvement obtained using data set L is less than that using data sets S and M when the number of LPs is 10. However, since data set L contains larger hub locations than data sets S and M, the *Split_Enhance* method becomes more effective in load balancing when the number of LPs increases. For example, with 200 LPs, the improvement using data set L is 68%, compared to 54% and 56% for data sets S and M respectively.

To verify the parallel efficiency, we calculate it using the following formula:

$$E_k = \frac{T_{10}}{T_k} \times \frac{10}{k}, \quad (6)$$

where k is the number of LPs, and T_k is the execution time of the simulation with k LPs. We use the execution time of the simulation with 10 LPs (T_{10}) as the baseline for measurement. Fig. 17 shows that the *Split_Enhance* method has better E_k than the *Original* method, especially when there are over 100 LPs. The parallel efficiency of both the *Original* and *Split_Enhance* methods drops with increasing number of LPs. However, the drop of the *Original* method is much faster, especially for data set L. In contrast, for the same data set, the parallel efficiency of the *Split_Enhance* method is always above 70%.

Fig. 17. Efficiency of speedup on NSCC cluster.

6 RELATED WORK

SCN models the interactions between people in real life. The main application of SCN is to study epidemic. It can be also applied to the simulation of other social processes such as information propagation [Apolloni et al. 2009] and terrorism [Moon and Carley 2007]. A SCN can be represented using either agent-to-agent (A2A) graph [Bisset et al. 2009] or agent-to-location (A2L) graph. Since A2A graph lacks the spatial and/or temporal information about contact, A2L graph is a more realistic representation. Several data sets using A2L graph have been studied or published [Bisset et al. 2006; Gallagher et al. 2017; Swarup et al. 2014]. They are synthesized by using census data (for population structure) and by combining microscopic survey with simulation (for dynamic activity schedule) [Xia et al. 2015].

The performance of SCN simulation has attracted continuous attentions from researchers due to the challenges as discussed in Section 2. EpiSimdemics [Barrett et al. 2008] is an epidemic simulation platform which supports over 100 million people. It has evolved to obtain great scalability with thousands of CPU cores [Bhatele et al. 2017; Yeom et al. 2014]. Especially, it addresses the problem caused by hub locations at application level by splitting them into small sub-locations, as we discussed in the introduction section. In contrast, we use a system-level decomposition which maintains the original semantic of the SCN.

Besides SCNs, many networks derived from real world are also skewed and can be modeled as scale-free networks where the node degrees follow a power law distribution [Barabási and Albert 1999]. Nguyen and Fujimoto [2016] proposed a link partitioning method for simulation over scale-free telecommunication network. In their approach, each link between routers instead of router itself is mapped onto an LP. In this way, different ports of a router can be simulated in parallel so that the load imbalance caused by hub nodes is avoided. However, this edge-oriented approach is not suitable for A2L graph, because the contact computing of location cannot be associated with each edge. In our approach, we also split hub nodes, but we still employ vertex-oriented processing (i.e., executing the program defined for the node of network) for simulation.

Vertex-cut is also used in the PowerGraph, a graph processing framework proposed by Gonzalez et al. [2012]. It has shown great performance improvement in machine learning applications due to the communication reduction and load balancing by vertex-cut. The same idea is also adopted by other graph computing related work [Pearce et al. 2013]. In their approaches, one edge is assigned to only one partition while a vertex can be split into parts as the result of the edge assignment. Master-slave pattern is used to manage the communication amongst splits. One split is selected as a master and others are as mirrors. However, in SCN simulation every split may contain carriers and the messages from carriers need to be duplicated at each split location. Hence, the peer-to-peer communication pattern has to be adopted in our approach.

In section 4.3, we have discussed the difference between agent decomposition proposed in this paper and agent duplication, for example, used by the agent-based simulation framework - Repast HPC [Collier and North 2013]. In Repast HPC, for an agent near a partition boundary, a full duplication of the agent will be created at the neighboring partition so that other agents at the neighboring partition can access its state locally. For the agent decomposition proposed in this paper, splits of an agent are created by decomposing the agent's activity list. Hence, each split of the agent has different data (i.e., activity list) and conducts its own activities locally, independent from other splits.

The load balancing in large-scale agent-based simulation has been extensively studied. Geographic or graph partitioning [Xu et al. 2014] can be conducted before simulation to obtain initial balanced partitions. A community-detection based method [Hou and Yao 2010] was also proposed to find clustered entities for partitioning so that the communication overhead can be minimized. Static approach assumes the workload will not change at runtime. Dynamic approach is used for changeable workload. A common way is to repartition and migrate simulated entities based on the execution profile [Alt and Wilsey 2014]. D'Angelo and Ferretti [2009] proposed a general framework supporting simulation over skewed networks. In their framework, the closeness relationship between agents can be discovered during run-time and these agents will then attempt to stay together by migrating across partitions. The load balancing is also achieved during this process. The method is suitable for dynamic system with light-weight agents. Petkova et al. [2016] used the label propagation algorithm (LPA) to detect community and re-partition network during simulation. In this paper, we improve network structure by node decomposition for better partitioning, and apply the existing partitioning tool to obtain static load balancing. To improve load balancing dynamically, we use off-loading and asynchronous stepping techniques instead of migrating simulation entities. Specifically, we offload some large entities to underloaded partitions just in time, and meanwhile reduce the idle time caused by load imbalance during synchronization by asynchronously stepping agents.

7 CONCLUSION

The skewed degree distribution is a natural feature of SCN. A few hub locations in SCN have high degrees and can cause severe load imbalance and large communication overhead for parallel agent-based simulation of large-scale SCN. This paper proposes a comprehensive solution to address the performance problems caused by the hub locations. The solution includes hub location decomposition, agent decomposition and execution enhancement using off-loading and asynchronous stepping techniques. Hub location decomposition reduces the skewness of the network and thus helps to generate more balanced initial partitions. Agent decomposition localizes agents' access to locations and hence reduces the amount of communication. In addition, off-loading and asynchronous stepping techniques further mitigate the performance impact caused by load imbalance during runtime. Experimental results demonstrate that our comprehensive solution can obtain significant execution time improvement compared with the original method without explicitly handling the hub locations. Furthermore, our solution also shows good scalability, especially with large data set.

In the future, we will extend our study in two directions. First, the proposed methods will be evaluated using more scenarios and applications. For epidemic simulation, public interventions (e.g., quarantine and vaccination) and agent's behavior (e.g., making a decision to stay home or go to hospital when getting infected) should be taken into account. Meanwhile, the population structure and daily activities may vary with time. In these cases, dynamic network structure (i.e., adding and removing nodes and edges during runtime) needs to be considered. In addition, we also plan to apply our approach to other simulations such as information diffusion and large-scale disaster evacuation in a city [Lewis et al. 2013]. To apply our approach to a new application, application characteristics need to be studied in order to

determine the decomposition of agents and locations and to identify potential for offloading and asynchronous stepping. Second, the multi-core and GPU platforms will be exploited to further improve the performance. Our initial attempt using multi-threading encountered scalability bottleneck when each MPI process uses more than 4 threads. Similar experience was also shared in [Yeom et al. 2014], which suggested using three threads at most in one LP. We do believe that the multi-threaded implementation can be enhanced further with several techniques (e.g., lock-free, sequential memory accessing and local load balancing). However, the implementation should also take various other factors (e.g., software libraries and hardware features) into considerations.

ACKNOWLEDGMENTS

Yulin Wu and Wentong Cai would like to acknowledge the funding support from the Singapore National Research Foundation under its Campus for Research Excellence And Technological Enterprise (CREATE) program.

REFERENCES

- Amine Abou-Rjeili and George Karypis. 2006. Multilevel algorithms for partitioning power-law graphs. In *Proceedings 20th IEEE International Parallel & Distributed Processing Symposium*. IEEE, 124–124.
- Aj Alt and Philip A Wilsey. 2014. Profile driven partitioning of parallel simulation models. In *Proceedings of the 2014 Winter Simulation Conference*. IEEE, 2750–2761.
- Andrea Apolloni, Karthik Channakeshava, Lisa Durbeck, Maleq Khan, Chris Kuhlman, Bryan Lewis, and Samarth Swarup. 2009. A study of information diffusion over a realistic social network model. In *International Conference on Computational Science and Engineering, 2009. CSE'09.*, Vol. 4. IEEE, 675–682.
- Albert-László Barabási and Réka Albert. 1999. Emergence of scaling in random networks. *science* 286, 5439 (1999), 509–512.
- Albert-László Barabási, Rka Albert, and Hawoong Jeong. 2000. Scale-free characteristics of random networks: the topology of the world-wide web. *Physica A Statistical Mechanics & Its Applications* 281, 1 (2000), 69–77.
- Christopher L Barrett, Keith R Bisset, Stephen G Eubank, Xizhou Feng, and Madhav V Marathe. 2008. EpiSimdemics: an efficient algorithm for simulating the spread of infectious disease over large realistic social networks. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*. ACM, Article 37.
- Chris L Barrett, Stephen G Eubank, and James P Smith. 2005. If smallpox strikes Portland... *Scientific American* 292, 3 (2005), 54–61.
- Abhinav Bhatle, Jae-Seung Yeom, Nikhil Jain, Chris J Kuhlman, Yarden Livnat, Keith R Bisset, Laxmikant V Kale, and Madhav V Marathe. 2017. Massively Parallel Simulations of Spread of Infectious Diseases over Realistic Social Networks. In *Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE, 689–694.
- K Bisset, K Atkins, CL Barrett, R Beckman, S Eubank, A Marathe, M Marathe, HS Mortveit, P Stretz, and VA Kumar. 2006. *Synthetic data products for societal infrastructures and proto-populations: Data Set 1.0*. Technical Report. NDSSL Technical Report.
- Keith Bisset, Jiangzhuo Chen, Chris J Kuhlman, VS Kumar, and Madhav V Marathe. 2011. Interaction-based HPC modeling of social, biological, and economic contagions over large networks. In *Proceedings of the 2011 Winter Simulation Conference*. IEEE, 2938–2952.
- Keith R Bisset, Jiangzhuo Chen, Suruchi Deodhar, Xizhou Feng, Yifei Ma, and Madhav V Marathe. 2014. Indemics: An interactive high-performance computing framework for data-intensive epidemic modeling. *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 24, 1 (2014), Article 4.
- Keith R Bisset, Jiangzhuo Chen, Xizhou Feng, VS Kumar, and Madhav V Marathe. 2009. EpiFast: a fast algorithm for large scale realistic epidemic simulations on distributed memory systems. In *Proceedings of the 23rd international conference on Supercomputing*. ACM, 430–439.
- Stephen P Borgatti, Ajay Mehra, Daniel J Brass, and Giuseppe Labianca. 2009. Network analysis in the social sciences. *science* 323, 5916 (2009), 892–895.
- Nicholson Collier and Michael North. 2013. Parallel agent-based simulation with Repast for High Performance Computing. *Simulation* 89, 10 (2013), 1215–1235.
- Gabriele D'Angelo and Stefano Ferretti. 2009. Simulation of scale-free networks. In *Proceedings of the 2nd International Conference on Simulation Tools and Techniques*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), Article 20.
- Stephen Eubank. 2005. Network based models of infectious disease spread. *Japanese journal of infectious diseases* 58, 6 (2005), S.
- Stephen Eubank, Hasan Guclu, VS Anil Kumar, Madhav V Marathe, Aravind Srinivasan, Zoltan Toroczkai, and Nan Wang. 2004. Modelling disease outbreaks in realistic urban social networks. *Nature* 429, 6988 (2004), 180–184.
- Richard M Fujimoto. 2016. Research challenges in parallel and distributed simulation. *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 26, 4 (2016), Article 22.
- Shannon Gallagher, Lee Richardson, Samuel L Ventura, and William F Eddy. 2017. SPEW: Synthetic Populations and Ecosystems of the World. *arXiv preprint arXiv:1701.02383* (2017).

- Yuanzheng Ge, Rongqing Meng, Zhidong Cao, Xiaogang Qiu, and Kedi Huang. 2014. Virtual city: An individual-based digital environment for human mobility and interactive behavior. *Simulation* 90, 8 (2014), 917–935.
- Joseph E. Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. 2012. PowerGraph: Distributed Graph-parallel Computation on Natural Graphs. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation (OSDI’12)*. USENIX Association, 17–30.
- John J Grefenstette, Shawn T Brown, Roni Rosenfeld, Jay DePasse, Nathan TB Stone, Phillip C Cooley, William D Wheaton, Alona Fyshe, David D Galloway, Anuroop Sriram, et al. 2013. FRED (A Framework for Reconstructing Epidemic Dynamics): an open-source software system for modeling infectious diseases and control strategies using census-based populations. *BMC public health* 13:940, 1 (2013), 1–14.
- Diansheng Guo. 2007. Visual analytics of spatial interaction patterns for pandemic decision support. *International Journal of Geographical Information Science* 21, 8 (2007), 859–877.
- Bonan Hou and Yiping Yao. 2010. Commpar: A community-based model partitioning approach for large-scale networked social dynamics simulation. In *2010 IEEE/ACM 14th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*. IEEE, 7–13.
- Jiangming Jin, Stephen John Turner, Bu-Sung Lee, Jianlong Zhong, and Bingsheng He. 2012. HPC Simulations of Information Propagation Over Social Networks. *Procedia Computer Science* 9 (2012), 292–301.
- Yu A Kuznetsov and C Piccardi. 1994. Bifurcation analysis of periodic SEIR and SIR epidemic models. *Journal of mathematical biology* 32, 2 (1994), 109–121.
- B Lewis, S Swarup, K Bisset, S Eubank, M Marathe, and C Barrett. 2013. A simulation environment for the dynamic evaluation of disaster preparedness policies and interventions. *Journal of Public Health Management & Practice* 19 Suppl 2 (2013), S42.
- Il-Chul Moon and Kathleen M Carley. 2007. Modeling and simulating terrorist networks in social and geospatial dimensions. *IEEE Intelligent Systems* 22, 5 (2007), 40–49.
- Vy Thuy Nguyen and Richard Fujimoto. 2016. Link Partitioning in Parallel Simulation of Scale-Free Networks. In *2016 IEEE/ACM 20th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*. IEEE, 77–84.
- Roger Pearce, Maya Gokhale, and Nancy M Amato. 2013. Scaling techniques for massive scale-free graphs in distributed (external) memory. In *2013 IEEE 27th International Symposium on Parallel & Distributed Processing (IPDPS)*. IEEE, 825–836.
- A. Petkova, C. Hughes, N. Deo, and M. Dimitrov. 2016. Accelerating the distributed simulations of agent-based models using community detection. In *IEEE International Conference on Computing Communication Technologies, Research, Innovation, and Vision for the Future (RIVF)*. IEEE, 25–30.
- Robert S Pienta and Richard M Fujimoto. 2013. On the parallel simulation of scale-free networks. In *Proceedings of the 1st ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*. ACM, 179–188.
- Natasa Pržulj, Derek G Corneil, and Igor Jurisica. 2004. Modeling interactome: scale-free or geometric? *Bioinformatics* 20, 18 (2004), 3508–3515.
- Samarth Swarup, Stephen G Eubank, and Madhav V Marathe. 2014. Computational epidemiology as a challenge domain for multiagent systems. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*. International Foundation for Autonomous Agents and Multiagent Systems, 1173–1176.
- Youzhong Wang, Daniel Zeng, Zhidong Cao, Yong Wang, Hongbin Song, and Xiaolong Zheng. 2011. The impact of community structure of social contact network on epidemic outbreak and effectiveness of non-pharmaceutical interventions. In *Pacific-Asia Workshop on Intelligence and Security Informatics*. Springer, 108–120.
- Yulin Wu, Xiangting Hou, Wen Jun Tan, Zengxiang Li, and Wentong Cai. 2017. Efficient Parallel Simulation over Social Contact Network with Skewed Degree Distribution. In *Proceedings of the 2017 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*. ACM, 65–75.
- Huadong Xia, Kalyani Nagaraj, Jiangzhuo Chen, and Madhav V Marathe. 2015. Synthesis of a high resolution social contact network for Delhi with application to pandemic planning. *Artificial intelligence in medicine* 65, 2 (2015), 113–130.
- Yadong Xu, Wentong Cai, Heiko Aydt, and Michael Lees. 2014. Efficient graph-based dynamic load-balancing for parallel large-scale agent-based traffic simulation. In *Proceedings of the 2014 Winter Simulation Conference*. IEEE, 3483–3494.
- Jae-Seung Yeom, Abhinav Bhatle, Keith Bisset, Eric Bohm, Abhishek Gupta, Laxmikant V Kale, Madhav Marathe, Dimitrios S Nikolopoulos, Martin Schulz, and Lukasz Wesolowski. 2014. Overcoming the scalability challenges of epidemic simulations on blue waters. In *2014 IEEE 28th International Parallel and Distributed Processing Symposium*. IEEE, 755–764.
- Haifeng Zhang, Jie Zhang, Changsong Zhou, Michael Small, and Binghong Wang. 2010. Hub nodes inhibit the outbreak of epidemic under voluntary vaccination. *New Journal of Physics* 12, 2 (2010), 023015.
- Mingxin Zhang, Rongqing Meng, and Alexander Verbraeck. 2015b. Including public transportation into a large-scale agent-based model for epidemic prediction and control. In *Proceedings of the Conference on Summer Computer Simulation*. Society for Computer Simulation International, 1–8.
- Yu Zhang, Xiaofei Liao, Hai Jin, and Geyong Min. 2015a. Resisting Skew-Accumulation for Time-Stepped Applications in the Cloud via Exploiting Parallelism. *IEEE Transactions on Cloud Computing* 3, 1 (2015), 54–65.
- Zhaoyang Zhang, Honggang Wang, Chonggang Wang, and Hua Fang. 2015c. Modeling Epidemics Spreading on Social Contact Networks. *IEEE Transactions on Emerging Topics in Computing* 3, 3 (2015), 410–419.
- Tao Zou, Guozhang Wang, Marcos Vaz Salles, David Bindel, Alan Demers, Johannes Gehrke, and Walker White. 2011. Making time-stepped applications tick in the cloud. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*. ACM, Article 20.