Kurve *et al. Complex Adaptive Systems Modeling* 2013, **1**:12
http://www.casmodeling.com/content/1/1/12

Complex Adaptive Systems Modeling
a SpringerOpen Journal

RESEARCH Open Access

An agent-based framework for performance modeling of an optimistic parallel discrete event simulator

Aditya Kurve[1*], Khashayar Kotobi[1] and George Kesidis[2]

*Correspondence: ack205@psu.edu
[1]EE Department, The Pennsylvania State University, University Park, PA 16802, USA
Full list of author information is available at the end of the article

Abstract

Purpose: The performance of an optimistic parallel discrete event simulator (PDES) in terms of the total simulation execution time of an experiment depends on a large set of variables. Many of them have a complex and generally unknown relationship with the simulation execution time. In this paper, we describe an agent-based performance model of a PDES kernel that is typically used to simulate large-sized complex networks on multiple processors or machines. The agent-based paradigm greatly simplifies the modeling of system dynamics by representing a component logical process (LP) as an autonomous agent that interacts with other LPs through event queues and also interacts with its environment which comprises the processor it resides on.

Method: We model the agents representing the LPs using a "base" class of an LP agent that allows us to use a generic behavioral model of an agent that can be extended further to model more details of LP behavior. The base class focuses only on the details that most likely influence the overall simulation execution time of the experiment.

Results: We apply this framework to study a local incentive based partitioning algorithm where each LP makes an informed local decision about its assignment to a processor, resulting in a system akin to a self organizing network. The agent-based model allows us to study the overall effect of the local incentive-based cost function on the simulation execution time of the experiment which we consider to be the global performance metric.

Conclusion: This work demonstrates the utility of agent-based approach in modeling a PDES kernel in order to evaluate the effects of a large number of variable factors such as the LP graph properties, load balancing criteria and others on the total simulation execution time of an experiment.

Keywords: Agent-based modeling, Parallel simulation, Self organizing system, Game theory, Load balancing

Background

Parallel and distributed simulation techniques allow us to create and run large scale models that also allow fine grained modeling of the simulated entities. For example, the rapid growth of large-scale communication networks, particularly the Internet and the on-line social networks it supports, has brought an increased emphasis on the need to model and simulate them in order to understand their macroscopic behavior and to design and test

defenses against network attacks e.g., denial-of-service and Border Gateway Routing Protocol (BGP) attacks (Sriram et al. 2006). It has been observed largely that in many cases the results of such experiments are affected significantly by the level of abstraction that is chosen. For example, (Chertov and Fahmy 2011) studied the need to model forwarding devices such as switches or routers and the sensitivity of the experimental results to their level of detail in the model. Parallel and distributed simulation distributes the simulation tasks across multiple machines and, in this way, exploits their combined resources. A challenge here is to maintain synchronization among collaborating machines in order to preserve the causality of processed events and at the same time attain a significant speed up.

Most parallel simulation mechanisms use the concept of a logical process (LP) which is a software component that runs in parallel with other LPs via processor time sharing or by running simultaneously on different processors in the case of multiprocessor systems. Each LP has a set of local variables that define its state. The state of the simulation at a given time is defined by the combined state of all LPs at that time. LPs communicate with each other using inter-process communication mechanisms such as shared memory or pipes. For the specific case of network simulation, a simulator kernel creates a distinct LP as a computational unit for each element of the simulated network model (Nicol and Fujimoto 1994). For example, all the properties and methods associated with a simulated router are executed and maintained locally in a distinct LP that represents it. This aids in the distribution of computation load across machines and also restricts the communication of an LP with only its "neighboring" set of LPs (*i.e.*, neighbours in the network topology under simulation). In this way, we can represent the LPs and their interdependencies with a graphical model.

Extensive studies have been conducted to optimize the performance of parallel simulations by focusing on different aspects such as methods of synchronization, effect of roll-backs, LP-to-machine assignment methods and so on. New methods that promise to speed up parallel simulations are evaluated directly on hardware platforms such as cluster computers. Although the performance of any proposed method should be benchmarked ultimately using a hardware platform, this work-flow also presents its own share of shortcomings. For instance, limitations posed due to available hardware and the simulated scenario in the experiment makes the method susceptible to fine-tuning to the hardware and the simulated experiment. An alternative is to create a computational or numerical model of the parallel simulator that factors in known dependencies (both deterministic and probabilistic) and then evaluate the proposed methods using this computational or numerical testbed. This allows us the additional luxury of testing the proposed methods under varying system environment variables since they are represented in software. The parameters that affect the total simulation time can be abstracted by choosing appropriate methods of representation. For instance, the particular scenario that is simulated can be represented as a graph of logical processes or LPs, and each LP is represented with its behavioral model of event generation and processing as shown in Figure (1). Note that other details can be ignored since they do not have consequence on the total simulation execution time of the experiment. The Discrete Event System Specification formalism (Zeigler et al. 2000) describes a hierarchical and modular method of system modeling where model of the specific scenario to be simulated is separated from the simulation engine using formal means of representing the simulated model as an

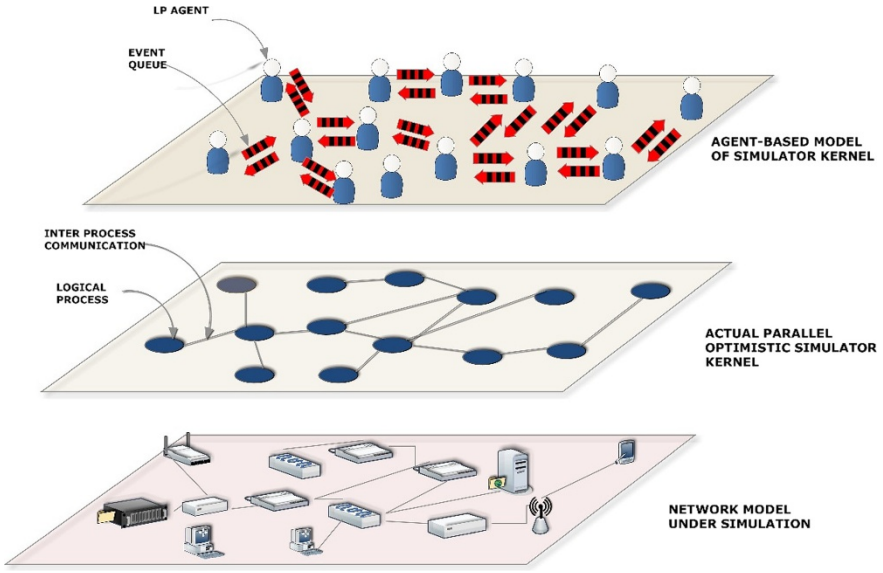

Figure 1 Network simulation illustration: Mapping to an agent-based model of PDES simulator.

input to the simulator. However since we are interested only in the simulation execution time of the experiment, we use a much simpler technique of representing the simulated scenario. One can extend the functionality of the LP agent class to include additional details.

In this work, we use an agent-based framework to simulate the behavior of a PDES (Parallel Discrete Event based Simulator) kernel wherein each LP is represented by an autonomous agent with its own set of distinct local variables. The agent-based modeling paradigm of LPs is novel and appropriate especially in the context of optimistic synchronization among LPs that requires each LP to maintain a local simulation state and use messages or events to communicate and synchronize with other LPs. Agent-based modeling allows us to study the resulting global performance metrics of the simulator kernel. The total time required for simulation is a non trivial and a complex (in many cases nondeterministic) function of various system parameters. The dynamics of an optimistic PDES kernel are easier to describe at the level of an LP in terms of the event generation and processing behavior. These local dynamics, their causal effects and the effect of state rollbacks have a direct consequence on the global performance metrics of the system, which in this case happens to be the total simulation wall-clock time of the experiment. We use this framework to evaluate an LP assignment scheme where each LP makes its own informed choice of a processor based on a novel incentive based cost criterion. The agent based modeling framework also makes it convenient to implement our assignment scheme since it is based on a local cost criterion at the level of LPs which are agents in our model. This is akin to a self organizing network of LP agents that form clusters across the machines.

Background

In the following subsections we detail some important background ideas and then develop our agent-based model of the simulator in the sections thereafter.

Need for parallel and distributed simulation

Networks in the form of social connections, information-carrying links, collaborative associations and reputation systems are ubiquitous. Large and complex networks result from the association of entities distinct in character, preferences, and inherent design. Network modeling is essential for understanding the behavior of a combined system of heterogeneous individual nodes that interact with each other. It is used, e.g, to test routing and forwarding protocols, defenses against network attacks and to study community structures in social networks. A typical communication network may contain tens of thousands of nodes with a large number of links. The nodes in reality can be whole routers, CPUs, or switches, individual router ports, firewalls, or end systems. (Sriram et al. 2006) studied the large scale impact of BGP peering session attacks that can cause cascading failures, permanent route oscillations or the gradually degrading behavior of the routers, using an Autonomous System (AS) level topology which was down-sampled from a typical AS-level topology consisting of 23,000 ASs and 96,000 BGP peering links (Zhang 2013). There were a fidelity/complexity tradeoff when simulating such a network. Some methods scaled down the network under study (Psounis et al. 2003) by omitting some intricate details and cleverly choosing the parameters that were expected to maximally affect the simulation results so as to minimize the loss of fidelity as much as possible. Recent papers such as (Gupta et al. 2008) introduced new methods to represent groups of nodes by a single node while (Carl and Kesidis 2008) studied path-preserving scaled-down topologies for large scale testing of routing protocols. Dimitropoulos et al. (2009) suggested use of annotations to a simple unweighted, undirected graph to represent the original network. However, the reliability of scaled-down methodologies depends largely on the assumption of low sensitivity of the outcome of the experiment to certain microscopic factors which may be at best crudely modeled by scale-down. An important macroscopic behavior of a network might be due (in an *a priori* unknown way) to a microscopic behavior ignored in the simulated model, thus resulting in inaccurate simulation results.

Parallel and distributed simulation: synchronization for causality

With the advancement of distributed processing systems, computing power was increased and one could thus feasibly represent networks using more refined models. Simultaneously, the theory of distributed simulation developed along with practical implementation of simulators such as PARSEC (Bagrodia et al. 1998). In the case of network simulation, one can represent each node in a network by a logical process (LP). An LP is an object that consists of a set of variables known as its "state" along with functions specific to the type of node being modeled. It normally has an event list containing time-stamped events to be executed. The local variables of an LP change when an event is processed. The value of these variables at any given time defines the state of the LP at that time. An LP maintains a local (simulation) time variable that contains the time stamp of the event currently being processed (if busy) or the most recent event processed (if idle). Events are stamped with their time of execution (in simulation time). Each LP maintains a list of events to be processed.

The methods to synchronize event execution between LPs running in parallel can be classified into two major types: conservative (Chandy KM, Misra 1981) and optimistic, e.g. Time Warp (Jefferson 1985). In the conservative methods, LPs strictly follow

time-causality, *i.e.*, all events are processed in the strict order of their time stamps. Each LP tries to ensure, before processing an event A, that no other event, say B, with time stamp less than that of A, will arrive at the LP subsequent to the processing of event A. This time causality is followed by assuming that the graph topology of LPs is fixed and known beforehand. LPs communicate via messages and each message-carrying link ensures that they are sent in the order of their time stamps. Synchronizing null messages are exchanged between LPs to assure each other that no event with time stamp less than a specified time will be sent. Optimistic synchronization allows for non-causal event execution. An LP is allowed to process events "ahead of time" without any kind of synchronization assurance. Each LP maintains its own local time and processes the event with lowest time stamp in the event list. If an LP receives an event time-stamped which is less than its local time, it rolls back in time to the time stamp of the event. Rolling back means restoring a state prior to the time stamp of the event that triggered the rollback. In order for it to rollback to a prior state, an LP should archive the past history of its states and events. The combined system of LPs maintains a global variable called *global-time* which is equal to the minimum local time across all the LPs. Hence the global time is indicative of the overall progress of the simulation.

Using agent-based framework for performance modeling of a PDES kernel

The system modeled using an agent-based framework is characterized or described in terms of the actions, behavior, beliefs, etc. of an individual autonomous agent in a group of large number of such autonomous agents. To a large extent, the paradigm of agent-based modeling can be said to use the bottom-up approach to characterize, represent, predict or recreate a system or a phenomenon. However, agents have been used to drive large scale software systems (Jennings 2001) where the architecture is specified using the top-down approach. A common underlying theme motivating agent based modeling is that most complex phenomena observed in the physical world are the consequence of a much simpler set of rules that govern the dynamics of a large number of constituent entities that it is comprised of. For example, the Axelrod's model of social dissemination (Axelrod 1997) tries to explain the consensus towards cultures and the simultaneous existence of different cultures in the society using a simple set of rules of interaction between individuals or behavioral models that explain the flocking of birds (Reynolds 1987) or ant colony optimization (Dorigo et al. 2006). Agent-based modeling has received a large amount of interest in the last decade even from the non-computing research communities in areas such as social sciences and ecology (Niazi and Hussain 2011). It is a comparatively new method of modeling and a large number of existing models can be extended under this paradigm. Borschev and Filippov (2004) illustrates how the classic predator-prey model can be enriched by an agent-based model that makes more realistic assumptions without any significant addition to the model complexity. A large number of agent-based modeling toolkits such as NetLogo Tissue and Wilensky (2004) and Repast (North et al. 2007) have been designed in the last decade. A useful comparative evaluation of the existing toolkits has been provided in (2007). A high-level approach to performance prediction of simulators has been done previously using probabilistic models for event thread generation. These statistical methods use analytical means instead of simulations to predict the performance or speed up of synchronized iterative algorithms on multiprocessors. For instance, Agrawal and Chakradhar (1992) uses maximum

order statistics on random variables that affect the speed-up on parallel machines. Xu and Chung (2004) proposes similar analytical methods for performance prediction of synchronous simulation. Agent-based modeling has always been explored as an alternative to equation-based modeling (Agent-based modeling vs. equation-based modeling). We argue that in the case of optimistic synchronization, analytical models are not feasible due to the complex and time-varying inter-dependencies of various parameters and hence we resort to agent-based approach. In the following sections we will demonstrate the application of agent-based modeling to performance modeling of an optimistic PDES kernel. The model is used to compute the simulation wall-clock time of the kernel for an experiment described by an event initialization list based on the dynamics and interactions at the level of LPs. The agent-based model can be used to evaluate and compare different cost criteria for LP assignment. In this work we focus on the additive cost framework described in (Kurve et al. 2011a) (See Appendix B) and the alternative cost framework (See Appendix C). This model can serve as a testbed to evaluate other cost criteria that can be more complex than the simple additive relationship in modeling effects of computation load imbalance and communication delays on the number of roll backs and consequently the simulation execution time.

LP assignment and simulation time

The total simulation time of the experiment is sensitive to the assignment of LPs to machines. In our model we focus on two aspects that are direct consequence of the LP assignment: the resulting load imbalance across the machines and the inter-processor communication[a]. We study the effect of optimization of these parameters on the total simulation time of the experiment. Our bi-criteria optimization algorithm assumes an additive relationship, *i.e.*, the total simulation time is represented mathematically as the addition of the load balancing cost and weighted inter-process communication cost. In the case of an optimistic PDES, both the parameters induce synchronization overheads in the form of rollback events that slows down the advance of the global simulation time. The graph partitioning problem is a well-known NP-complete problem that considers both these aspects. It is formally defined as follows.

Let $G = (V, E)$ be an undirected graph where V is the set of nodes and E is the set of edges. Suppose the nodes and the edges are weighted. Let w_i represent the weight of the i^{th} node and let c_{ij} represent the weight of the undirected edge $\{i, j\}$. Then the K-way graph partitioning problem aims to find K subsets $V_1, V_2, .., V_K$ such that $V_i \cap V_j = \emptyset \ \forall \ i, j$ and $\bigcup_{i=1}^{K} V_i = V$, $\sum_{j \in V_i} w_j = \frac{\sum_k w_k}{K} \ \forall \ j$ and with the sum of the weights of edges whose incident vertices belong to different subsets minimized.

Heuristics to solve the graph partitioning problem primarily make use of spectral bisection methods (Pothen et al. 1990) or multilevel coarsening and refinement techniques (Karapis and Kumar 1996). Spectral bisection methods calculate the second smallest eigenvector, known as Fiedler vector, of the modified adjacency matrix of the graph. These methods by far give the best results. However, finding the Fiedler vector is computationally very expensive. For "geometric" graphs in which coordinates are associated with the vertices, geometric methods are available which are randomized and are quicker than spectral bisection methods. Multilevel partitioning algorithms are by far

the most popular techniques. All of these discussed methods use centralized computational implementations involving access to global state information. A periodic refresh of partition is needed in most cases because of the highly time varying nature of the load generated across different LPs. The exchange of a chunk of nodes that are identified using a sparse cut metric was studied in Kurve et al. (2011b). In our work (Kurve et al. 2011a), we present an alternative to this, where decisions are made at the LP or node level. We model a game theoretic framework where the local cost for each LP incentivizes the LP to choose a processor so as to minimize the total simulation time of the experiment (represented by the "social welfare"). LPs can make distributed decisions eliminating the need for a separate software or hardware resource to refresh the partition across the machines. We are particularly interested in the class of games known as "potential games" (Monderer and Shapley 1996) which guarantee a descent in a global cost function (indicative of the system performance) with each decision made at the local (machine or node) level. The global state required to make an accurate choice of LP is independent of the total number of LPs and scales linearly with the number of neighbors of an LP. Please refer to Appendix A and B for more details on the local cost criteria. Note that the local incentive based criteria can be applied in other scenarios which involves the two competing incentives for load balancing and clustering. For instance, Kurve et al. (2013) has studied the application of the local incentive criteria in super-peer based peer-to-peer (P2P) networks.

Method

We designed the model using Netlogo (Tissue and Wilensky 2004). Our model is based on simulating the LPs as agents with individual characteristics and then observing the system level performance of the combined graph of nodes in terms of the simulation execution time. Some of the key features of the model are listed below:

1. We do not consider any specific simulation scenario and both the LP agent class and the class of events are generic to accommodate different particularities. The LPs can represent a router or a switch in the case of network simulation or logic gates in gate-level simulation of VLSI circuits. We describe the LP agent and event classes in detail in the following subsections.
2. One of the inputs to the model is a list of events to which event lists of the LPs is initialized before the simulation begins.
3. We abstract the simulation model across different scenarios using a graph of LPs and an event generation model defined by the initial event list and the cause-effect relationships between different events resulting in the event execution thread across LPs during simulation.
4. The parallel processing hardware is modeled as simulated artifact in terms of the number of processors, relative speed of each processor and the mean intra and inter processor communication delays.
5. We focus our attention on the total simulation time and the synchronization overhead in terms of the number of rollbacks, which depends largely on the event generation and processing model rather than an actual specification of the simulation scenario.

To illustrate the abstraction mechanism, consider an oversimplified scenario of N peers sharing content using a peer-to-peer (P2P) system such as Gnutella. We would like to simulate the query generation and resolution mechanism in the system. Suppose the queries are resolved using random walk over the overlay network graph. Using our method of abstraction, we can represent this using a graph of LPs which is similar to the graph of the overlay network. Each peer is represented by a unique LP in the graph. The events for each LP correspond to queries generated by the LP itself and those forwarded by its neighbors. In this case the event list for an LP is initialized based on the query generation rate of each peer. The query forwarding mechanism is represented using limited hop event forwarding and time to process each event. Note that our abstraction is agnostic about the actual changes that happen to a local state of an LP (representing a peer) in the simulation, because this does not directly affect the total simulation time.

As shown in Figure 2, the inputs to the model are the graph of LPs, the initial event list for each LP and the LP assignment algorithm and the output is the simulation execution time.

The LP agent class

We now describe some important properties and methods that define the base class of an LP agent. The LP agent base class is based on optimistic synchronization and uses rollback to synchronize its event execution sequence. It supports two basic event types: regular and rollback. One can easily derive more specific classes of LP agents that support a richer class of events. An LP has a set of local variables which are specific to the scenario under simulation. We do not factor in these scenario-specific local variables in the total simulation time and hence in our base LP agent class, this is an empty set. It has an *event-list* that holds the list of pending events that it needs to process. Each event has an associated time stamp and the agent processes each event in the order of the time stamps starting with the least. The variable *local-time* is equal to the time stamp of event being processed currently if the agent is busy or the time stamp of last processed event if not busy. When *busy?* is true, the agent performs no operation and simply decrements the *busy-tick* timer until it becomes zero. It also contains a list of event history with the local state of the LP archived before processing each event. In the case of a rollback event or a non-causal event (event whose time stamp is less than the local simulation time), the state of the LP is restored back to the time stamp of the non-causal event using the archived states from the *event-list-history*. Tables 1 and 2 shows a list of some important properties and member functions of the LP agent base class.

The *process_causal_event()* member function in the base class simply loads the *busy-tick* counter and sets the *busy?* bit true and possibly generates events in the neighboring LPs when *busy-tick* becomes zero. This is because we are simply interested in the time

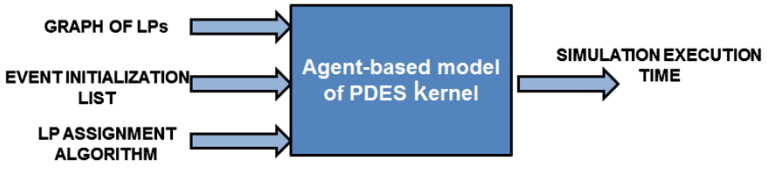

Figure 2 Software based model of a simulator.

Table 1 Properties of an LP agent base class

local-time	Contains time stamp of event being processed
state	Local variables of the LP
event-list	List containing pending events
event-list-history	List containing the archived local state of LP before each processed event in the past
busy-tick	The time remaining in wall-clock units to finish processing the current event
busy?	A boolean variable indicating if the LP is busy processing an event or idle.
my-machine	Current machine assigned to the LP
sim-time	Time in wall-clock ticks needed for processing the current event

of execution of the events. As shown in algorithms 1 and 2 the difference between the two member functions that process non-causal and rollback events is that in the case of former, the LP state restoration is followed by the execution of the current event. Whenever an LP rolls back in simulation time, it sends rollback events to all its neighboring LPs that it had sent events in the past until the time of rollback. If both the regular event and its corresponding rollback event are present in the *event-list* of an LP then they nullify each other and both are deleted. Algorithm 3 shows the pseudo code for the *simulate()* member function which is called for every tick of wall-clock time.

Event generation and processing model

The properties of the base event class are listed in Table 3.

In our model, each thread of events has a unique number denoted by *event-thread-number*. We define thread as a sequence of events where one event is generated by a previous event in the sequence. Each event in the event initialization list starts its own thread of events across LPs. This helps us track the spread of events in a limited-scope flooded packet-flows scenario. The time stamp of the events are stored in the *event-time* variable. The time stamp is the simulation time when the event is supposed to occur. The *event-type* variable tells the LP what its reaction to the event should be. Generally, there is a specified procedure call for every type of event. A rollback event is one of the types by default. In order to model delay in communication (in wall-clock time) between two LPs, we use the variable *event-tick*, which is programmed to a value by the LP that generates the event. At every tick of wall-clock time, *event-tick* is decremented by one unit until it reaches zero. An event can be processed by an LP only if its corresponding *event-tick* variable is zero. The variables *event-list-history* are the lists containing the information the

Table 2 Member functions of an LP agent base class

process_causal_event()	Member function to process causal event
process_noncausal_event()	Member function to process noncausal event
process_rollback_event()	Member function to process rollback event
generate_event(agent-list)	Generate new events for the thread of currently processing event in each of the agents in *agent-list*
create_neighbor_list ()	Returns a agent-list which is a subset of neighbors
simulate()	Member function called at every tick of wall-clock time
get_busy_time()	Returns the time needed to process the current event. Depends on the type of event, current processor load and processor speed
compute_node_weight()	Computes node weight
compute_edge_weight()	Computes the weight of edges with all its neighbors

Algorithm 1: *process_noncausal_event()*

```
for each event in event-list-history
 {
   if event-time > time of current-event then
      Restore event data event-list, event-time, event-type, event-tick, event-count.
       Delete event data from event history lists
       for each neighboring node
        {
          if current-event exists in event-list of neighbor
             Delete the event
          else if current-event exists in event-list-history of neighbor
             Send rollback event to the neighbor
          end if
        }
   end if
 }
Put all data of current-event in event history lists
Call process_causal_event() for the current-event
```

events already processed by the LP along with the contemporary state of the LP before the event was processed. In the case of a rollback, the LP restores its prior state from these variables. The LP regularly flushes out the data of events from the history with time stamps less than the global time. This is because the LP will never have to rollback to a state prior to the global time. Note that in such a model of the simulator, the functions that actually process events are generic. All that is needed is the time in wall-clock ticks stored in the variable *sim-time*, required for processing the event and the neighboring nodes where new events will be created as a result of processing this event. The *sim-time* is a function of the speed of the machine on which the LP resides given by *my-machine*, which in turn depends on the number of LPs that reside on that machine. We use a simple limited scope event forwarding using the generic framework of the simulator model. In this model, events are generated at random times by randomly chosen LPs and these events traverse the network for a limited number of *hops*, *i.e.*, each node that receives such a packet forwards it to a randomly chosen neighbor, provided the hop count given by *event-hop-count* is not zero. In our experiments, we randomly initialize the *event-list* of each LP that generate such a thread of events in such a way that the load generated

Table 3 Properties of an event base class

event-thread-number	Thread number of the event
event-time	Time stamp of the event
event-type	Type of event: regular or rollback
event-tick	Waiting time in wall-clock units the event is ready to be processed. This is used to model communication delays between LPs
event-hop-count	The maximum hop count of the event thread. Event threads survive for limited hops equal to this value

across the LPs becomes highly dynamic during the course of simulation. More specifically, we generate "hot spots" of traffic or a cluster of LPs that generate large amounts of traffic over a short period of (simulation) time. The locations of these hot spots change regularly.

LP assignment algorithm

Initial static LP-to-Machine assignment

We argue that due to initial uncertainty of computation and communication costs and their dynamic nature, an optimum partition is not possible *a priori*. As a result, we employ a simple initial partitioning method in which each machine chooses an initial "focal node" from among the nodes of the graph and then expands hop-by-hop to include neighboring nodes. The idea is to have connected sub-graphs within each partition so as to minimize inter-process communication. To avoid contention between partitions, we require that each machine wait a random amount of time after every hop and check for a semaphore before claiming ownership of new nodes. Unit edge and node weights are assumed during initial partitioning. The choice of the focal nodes is important to ensure a high probability of a good initial partition. As will be described later, the iterative partition refinement algorithm converges to a local minimum. Hence, a good initial partition might improve the chances of converging to the global minimum or a good local minimum. Refer to Appendix A for additional details.

Iterative partition refinement

The partition refinement algorithm utilizes the cost framework developed in the game theoretic study. During the course of the simulation, each LP can decide to switch processors either synchronously or asynchronously. In synchronous transfers which we refer to as a refinement step, LPs take turns to evaluate their costs and the prospective new machine based on the cost framework. In asynchronous transfers, LPs decide to switch without a global synchronization scheme. However, asynchronous transfers might lead to

Algorithm 2: *process_rollback_event()*

```
for each event in event-list-history
 {
   if event-time > time of current-event then
      Restore event data event-list, event-time, event-type, event-tick, event-count.
       Delete event data from event history lists
       for each neighboring node
        {
          if current-event exists in event-list of neighbor
             Delete the event
          else if current-event exists in event-list-history of neighbor
             Send rollback event to the neighbor
          end if
        }
   end if
 }
```

Algorithm 3: *Simulate()*

```
Remove events from event-list-history with event-time < global-time
Remove rollback events from event-list-history if event-time > local-time
if busy? = idle then
   current-event = event in event-list with lowest time stamp and event-tick = 0
   if current-event ≠ rollback and time stamp of current-event ≥ local-time then
      busy? = true
      busy-time = get_busy_time()
       Put all data of current-event in event history lists
   else if current-event ≠ rollback and time stamp of current-event < local-time
       process_noncausal_event()
   else if current-event = rollback then
       process_rollback_event()
   end if
 else
    busy-time = busy-time - 1
    if busy-time = 0
       busy? = false
       if event-count of current-event ≠ 0 then
          generate_event(create_neighbor_list ())
       end if
    end if
 end if
for each event in event-list
 {
    if event-tick ≠ 0 then
    event-tick = event-tick - 1
    end if
 }
```

inconsistent decisions due to the likelihood of simultaneous transfers of more than one LPs. Note that the inconsistencies can be eliminated if the transfers occur between two different pairs of machines. We thus assume the use of a processor *mutex* in the case of asynchronous transfers that allows only single transfer for every processor. The transfer rates in our model are controlled by the global parameter *partition-refine-freq* which determines the frequency when the the node weights corresponding to the computation cost of each LP and the edge weights corresponding to the communication cost between LPs are recomputed and the LPs are triggered to evaluate processor transfers.

In the case of synchronous transfers, LPs are triggered to compute their "destination processor" based on the current state of the system. Then the "most dissatisfied" LP in the system is allowed to make transfer to its destination machine. The "most dissatisfied" node is the one which would benefit the most in terms of its cost by changing its machine. The cost function used here can be either of the two defined in (3) or (6). In our experiments we make comparative evaluation between the two cost frameworks. Hence $C_i(k)$

Algorithm 4: Simulator engine pseudocode for limited scope flooded packet flow

```
Initialize global variables.
ask nodes
{
  Initialize local variables
  Initialize the event-list
  initial-partition()
}
wall-clock-time = 0
while event-list of all nodes not empty do
wall-clock-time = wall-clock-time + 1
ask nodes
{
simulate()
}
global-time = minimum local-time and time stamps of events in event-list across all
nodes
if remainder sim-time/partition-refine-freq= 0
ask nodes
{
 compute_node_weight()
 compute_edge_weight()
}
refine_partition
end if
end while
```

represents the cost of the i^{th} node if it were to be assigned to the k^{th} machine and $C_i(r_i)$ represents the current cost of the i^{th} node. We define the dissatisfaction of the i^{th} node as

$$\Im(i) = C_i(r_i) - \min_k C_i(k) \tag{1}$$

Hence the most dissatisfied node is the one with maximum value of $\Im$. The most dissatisfied nodes in the system having $\Im = 0$ indicates that the algorithm has converged to a local minimum. We can prove that this algorithm converges to a local optimum of the social welfare function (Kurve et al. 2011a).

Simulation engine

Finally, the pseudocode for the entire simulation engine of the agent-based model is shown in algorithm 4. It begins with the initialization of the global and local variables including the event-list of all the LP agents. At each tick of *wall-clock-time*, *simulate()* member function for all the LP-agents is invoked exactly once. The simulation is continued and the *sim-time* is incremented until all the LP agents have an empty *event-list*. *partition-refine-freq* defines how frequently the LP assignment algorithm is invoked. Particularly, it is the interval between two invocations of the LP assignment algorithm. Before every invocation of the LP assignment algorithm, the node and edge weights are

recomputed. *compute_node_weight()* and *compute_edge_weight()* are used to estimate the computation cost of an LP and the communication cost over the edge with its neighboring LPs. There can be many different ways of defining these functions. In our experiments, we use the length of the *event-list* as being indicative of the computation cost that the LP will generate in the near future. Hence, node weight is calculated as the length of a weighted *event-list* where each event in *event-list* is weighed by *event-weight*, *i.e.*, with a weight which is the number of machine cycles needed to process the event. Communication cost should quantify the amount of "event traffic" between two LPs. We estimate edge weight from the history of event traffic across the edge. Specifically, we calculate the number of events generated across the edge during a fixed interval of time given by *estimation-window*. We then multiply it by 100 to make its magnitude comparable to the computation cost and then assign it to the weight of the link.

Results and discussion

We evaluated the iterative LP assignment algorithms on the agent based model described above to compare the speed up in the simulation of an experiment. We also observed the performance under different settings of system such as synchronous and asynchronous LP transfers and for two different models of random graph generation. As described previously, one of the inputs to the model was a graph of LPs. The first model of random graphs that we used to generate an instance of the LP graph was the preferential-attachment model which is scale-free and, according to some studies, can be used to model the Internet topology at the level of Autonomous System (AS) (Bu and Towsley 2002). The second model incorporates geometric information along with the degree of a node, *i.e.*, each node has associated coordinates in two dimensions: while forming links, each node randomly chooses another from among a set of 15 closest nodes (in terms of distance with respect to the coordinate axes). Our NetLogo based model of LP graph consisted of 300 LP nodes generated from one of the models above each of which was an instance of the LP agent class described in Section 3.1. The event initialization list was created keeping in mind the highly dynamic nature of the load generated during a network simulation, *i.e.*, we created two genres of event threads: one that was uniformly distributed across all nodes with a relatively large value of *event-hop-count* and second one that had shorter hop count and was localized to a closely connected set of nodes. The former created a wider scope of event interdependencies between nodes that were far apart in geodesic distance while the later generated hot spots of traffic that were localized in space and time.

get_busy_time() as listed in Table 2 gave the total time in wall-clock units needed to process the *current-event*. We used the following relationship to compute the *busy-time*.

$$busy{-}time = \frac{weight\ \text{of}\ current - event \times Number\ of\ LPs\ sharing\ same\ processor \times 100}{processor - speed}$$

Note that the *processor-speed* is the normalized speed of the processor that the LP resides on. We set the inter processor communication delay as 10000 wall-clock ticks uniformly across all the links between processors, *i.e.*, any event that is created by an LP in another LP that resides in a different processor other than its own processor would expe-

rience a delay of 10000 wall-clock ticks. We set the value of *estimation-window* described in the previous section to 10000 clock ticks.

In our first set of experiments, we tuned the value of μ (relative weight given to inter-machine potential rollback-delay cost), *i.e.*, we tried to optimize the total simulation execution time of the experiment over different values of μ. The search space for μ is unbounded and hence we exponentially increased μ by powers of 10. For each setting, we kept all other parameters of the model constant, *i.e.*, the LP graph, initial partition before the simulation and sources of randomness such as event weights. We repeated this for ten different realizations of the LP graph and event thread initialization, and computed the average simulation time. As seen in Figure 3, the optimum values of μ for the two cost frameworks are different: about 100 for the first method and about 1000 for the second method. We can also observe that increasing the value of μ beyond a certain limit does not change the total simulation execution time.

The above experiment was performed for the geometric model of LP graph. In our next set of experiments, we further fine tune the value of μ to optimize the simulation execution time. In this case we varied μ from 0.1 to 1000 with an exponent of 2 and observed the optimal value of μ for the first method to be between 6.4 to 25.6, whereas for the second method it was between 400 and 800. However, as seen in Figure 4, the curves are not monotonous suggesting the presence of a large number of locally optimum points.

We also plotted similar curves using preferential attachment model to realize the random LP graph. Considering Figure 5, we observe quite a few differences from the graph in Figure 3. Most notable among them is the fact that the optimum values of μ for the two methods are different. This suggests that the additive rule that we used in both the cost frameworks, and the technique to estimate the node and edge weights, renders the efficacy of the partitioning algorithm sensitive to the properties of the LP graph. This is an important observation in the context of assigning the value of μ. Secondly we observe that the simulation execution-time rapidly increases as we increase μ beyond its optimum value unlike in the case of geometric model where it tends to stagnate after some value of μ. Similar to Figure 4, we plotted curves for the preferential attachment model

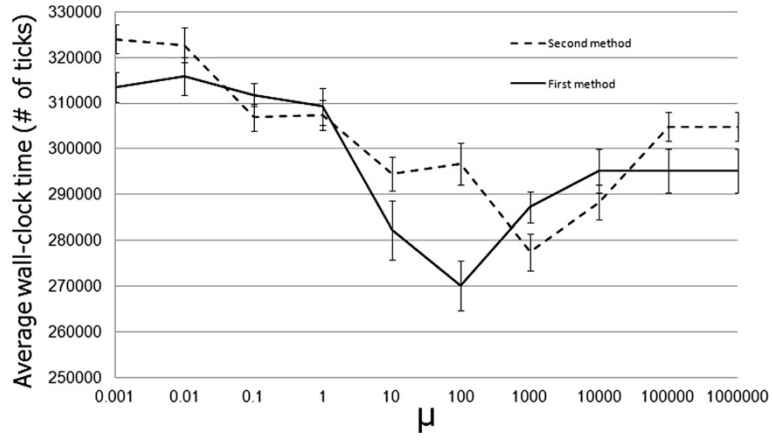

Figure 3 Finding optimum balance between computation and communication costs: For geometric model of random graph.

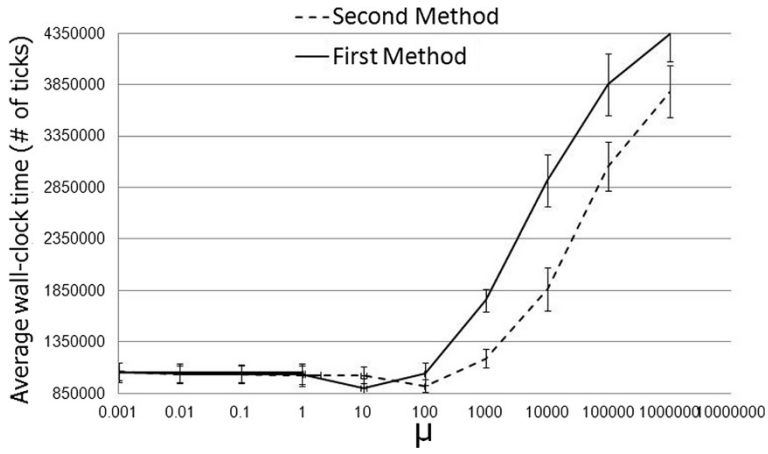

Figure 4 Resolving μ further: For geometric model of random graph.

in Figure 6. We can see that at this scale of resolution of μ, it is difficult to find the exact value of optimal μ due to the presence of large number of local minimum points in the curve. In all these cases it was observed that the first method generally performed better than the second.

In Figures 7 and 8, we plot the effect of the partition refinement interval on the simulation execution time. We observed that there is no significant gain in reducing the refinement interval below 5000 wall-clock ticks for both the LP graph generation models. However, this value depends on several different factors such as the dynamics associated with the events generated and the processing time of an event by the LP. These graphs further reiterate the need for partition refreshing in the case of a dynamic network simulation and that a static partitioning scheme is highly sub-optimal in this case.

From Figures 9 and 10, we can observe the effect of partition refinement interval on the distribution of rollback counts. We plotted the number of rollback events across five processors with the progress of simulation execution. We can observe that partition

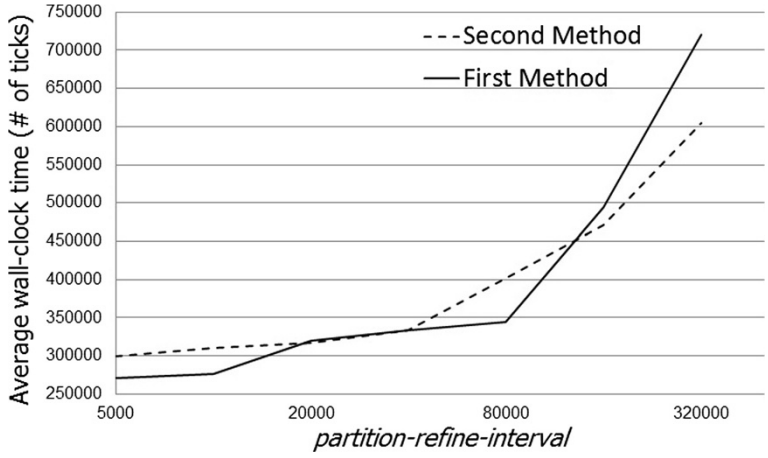

Figure 5 Finding optimum balance between computation and communication costs: For preferential attachment model of random graph.

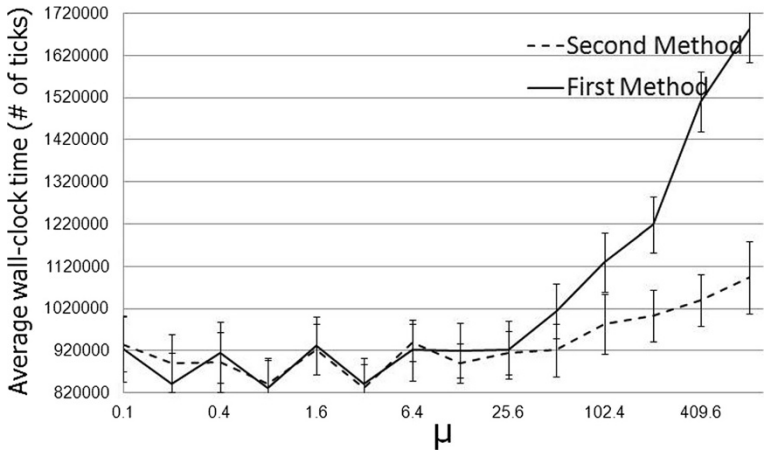

Figure 6 Resolving μ further: For preferential attachment model of random graph.

refinement reduces the variance of the number of rollback events across the processors as seen from Figure 10 when the interval is 20000 clock ticks as compared to Figure 9 when the interval is 100000 clock ticks.

In order to study the scalability of our results to variations in the LP graph size, we also plotted curves similar to Figure 5 in Figures 11 and 12 wherein the number of LP nodes in the graph were 100 and 500, respectively . We observed similar trends as in Figure 5 in both cases.

Conclusion

We presented a method of agent-0based modeling of an optimistic parallel discrete-event simulator. The agent-based modeling paradigm is novel to the study of a PDES performance and greatly expedites the evaluation of performance optimization techniques with different system settings. The model was used to evaluate and compare two LP-to-Machine assignment algorithms. In this process, we got several insights into the

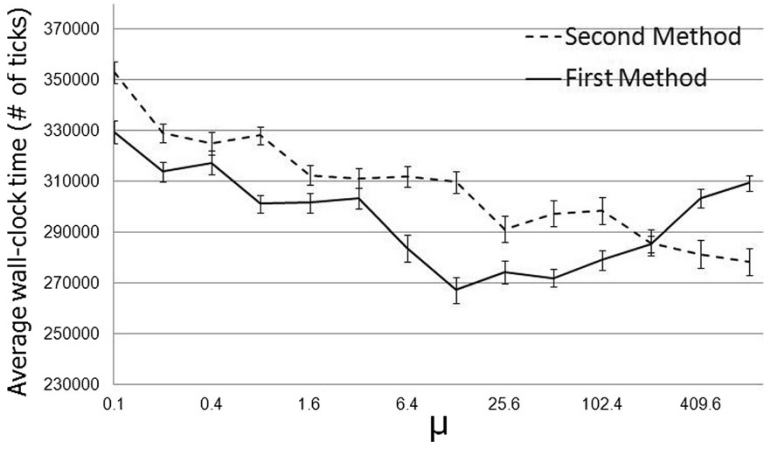

Figure 7 Effect of partition refinement frequency on simulation time: For geometric model of random graph.

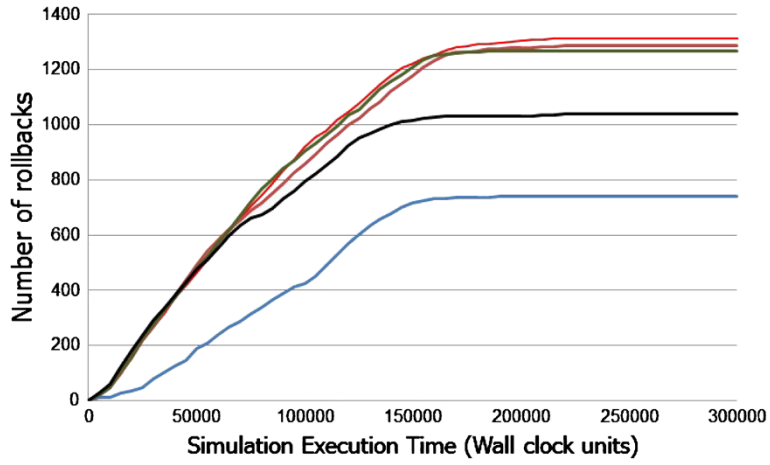

Figure 8 Effect of partition refinement frequency on simulation time: For preferential attachment model of random graph.

complex relationship between the LP agents and the simulation time. We observed that the comparative weight given to computation and communication delay cost in the LP assignment scheme depends on the properties of the LP graph. We studied the sensitivity of the simulator performance to other parameters such as partition refinement frequency, and the graphical properties of the network under simulation. The need for dynamic load balancing was further emphasized from the results of our experiments by studying the distribution of the rollback count curves with the advance of simulation execution time. One can use the proposed agent based modeling framework to study different LP to machine assignment techniques that characterize system performance using different objective functions. This framework can also be extended to study different rollback strategies and methods to estimate LP load and communication patterns based on the knowledge of current and past events.

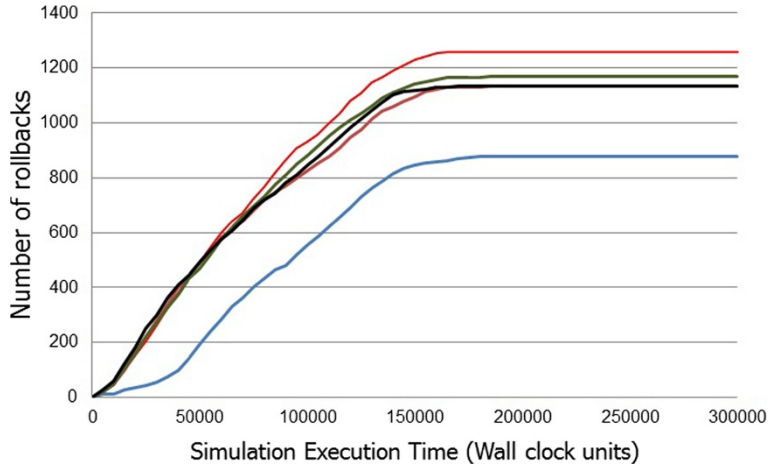

Figure 9 Rollback event distribution across processors with refresh interval of 100000.

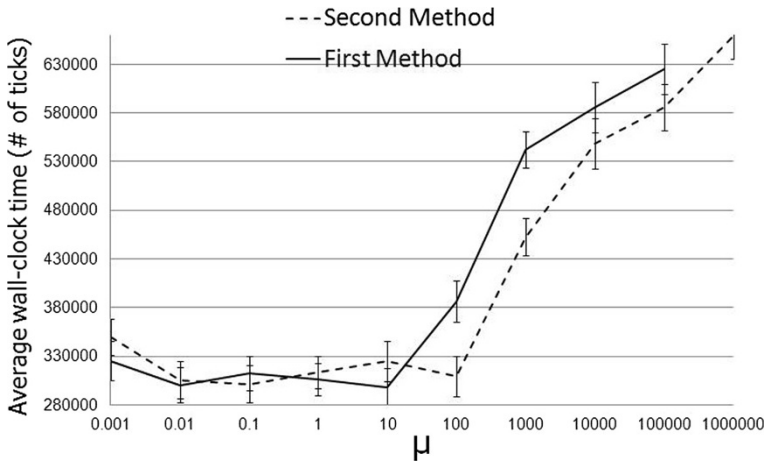

Figure 10 Rollback event distribution across processors with refresh interval of 20000.

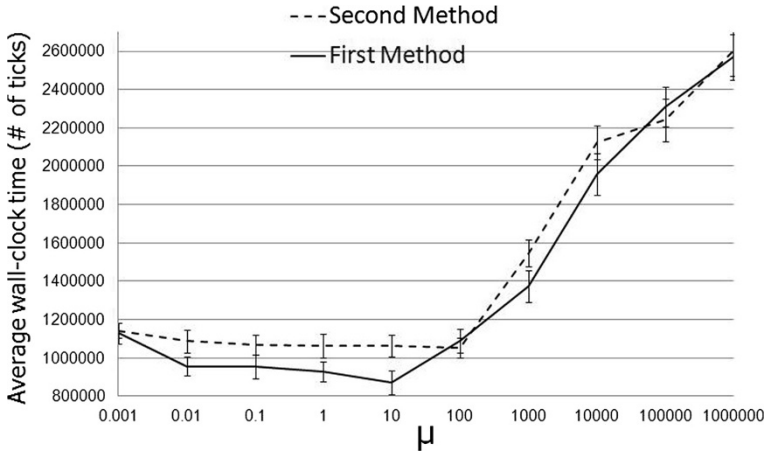

Figure 11 Finding optimum balance between computation and communication costs: For preferential attachment model of random graph of 100 nodes.

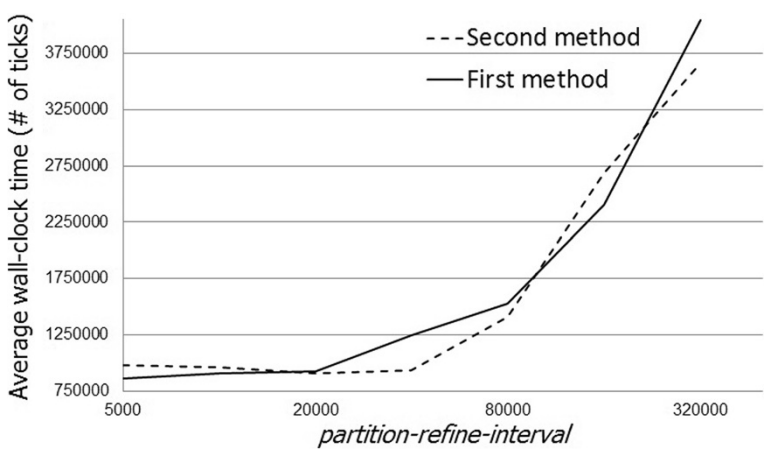

Figure 12 Finding optimum balance between computation and communication costs: For preferential attachment model of random graph of 500 nodes.

Appendix

Appendix A: initial partitioning algorithm

In this appendix, we describe the initial partitioning algorithm. As mentioned previously, the choice of focal nodes is important to get a good initial partition. The focal nodes are chosen so that they are at maximum geodesic distance from each other. Ideally we should find focal nodes such that each one of them is at least $2N_{\frac{|V|}{K}}$ geodesic distance away from the others, where $N_{\frac{|V|}{K}}$ is the mean number of hops that cover $\frac{|V|}{K}$ nodes. In this case, we should know the properties of the underlying graph for calculating $N_{\frac{|V|}{K}}$. We find this value for the example of an Erdos-Renyi random graph model.

Theorem 1. *For an Erdos-Renyi random graph $G = (V, E)$ with p as the probability of existence of an edge between any two nodes, if N_k and N_{k-1} are the expected number of nodes collected into a cluster at the k^{th} hop and $(k-1)^{\text{th}}$ hop respectively, then the expected number of nodes in the cluster by the $(k+1)^{\text{th}}$ hop is:*

$$N_{k+1} = N_k + (|V| - N_k)(1 - (1-p)^{N_k - N_{k-1}}) fork > 0,$$
$$N_{k+1} = 1 fork = 0.$$

Proof. Suppose after the k^{th} hop, the graph is divided into two sets of nodes: the set A of nodes obtained by the k^{th} hop and set A' of nodes not yet obtained. Let $|A| = N_k$ and $|A'| = |V| - N_k$. Denote the set of newly acquired nodes in the k^{th} hop by B, where $|B| = N_k - N_{k-1}$.

For any node $a \in A'$:

$$\mathrm{P}(a \text{ is not connected to any node in } B) = (1-p)^{N_k - N_{k-1}} \text{ and}$$
$$\mathrm{P}(a \text{ is connected to at least one node in } B) = 1 - (1-p)^{N_k - N_{k-1}}.$$

The number of nodes acquired during the $(k+1)^{\text{th}}$ hop is binomial, with $(|V| - N_k)$ trials and the probability of success being $1 - (1-p)^{N_k - N_{k-1}}$. Hence the expected number of nodes acquired during $(k+1)^{\text{th}}$ hop is $(|V| - N_k)(1 - (1-p)^{N_k - N_{k-1}})$ □

More practically, we would like to have the focal nodes as far as possible (in geodesic distance) from each other. This will ensure that the partitions formed after hop-by-hop expansion are somewhat equal in number of nodes per partition. Suppose $F = \{f_1, f_2, ..., f_K\}$ is the set of focal nodes. Hence we would like to have

$$F = \underset{H \subseteq V \text{ s.t. } |H| = K}{\arg\max} \; \min_{h,l \in H : l \neq h} d_G(h, l), \tag{2}$$

where d_G is the geodesic distance between the two nodes. We can attempt to find such nodes using heuristics. For example, start by assigning an arbitrary set of distinct nodes to each machine. In round-robin fashion, each machine takes a turn at finding a node from the set of nodes that are neighbors of its current focal node that increases the minimum of its geodesic distance with focal nodes of other machines. This becomes the new focal node for that machine. This process is iterated until there is no further improvement possible. We iterate this process over multiple initializations of the focal node set and the best set of focal nodes is identified. In the next phase, starting at the focal nodes, the

partitions try to collect-in nodes in their neighborhood, thus expanding their clusters. We can use random waiting time between two successive hops and semaphores to deal with contention issues, *i.e.*, when two or more machines try to claim ownership of the same node.

Appendix B: a local cost function

We will now describe our iterative local incentive for the partitioning algorithm (Kurve et al. 2011a) in detail. Consider an undirected weighted, graph $G = (V, E)$ representing the network model to be simulated. As described previously, the graph can be interpreted as a model of a network of LPs as vertices or nodes, with weights assigned to the nodes and edges of G. Thus, we want to first estimate the computational load generated by each node and the amount of communication between the logical processes associated with each node to assign node weights and edge weights, respectively, to the graph. Second, we wish to find a distributed technique to equitably load-balance among the machines while also taking into consideration the amount of inter-machine communication, the latter reflecting the risk of rollback. We will address the first problem in our following sections and focus on the second problem now.

- Let K be the number of machines ($K < |V|$). The graph G is partitioned among K machines or less, since in some cases where the cost of inter-processor communication is high, partitioning the workload among less than K machines might be optimal, *i.e.*, some machines may be not be assigned any LPs.
- Let b_i represent the computational load of i^{th} LP.
- Let c_{ij} denote the cost of communicating over the edge $\{i, j\}$, representing the average amount of traffic between node i^{th} and j^{th} LP .
- Let $r_i \in \{1, 2, ...K\}$ be the partition chosen by the i^{th} LP.
- Let the normalized capacity or speed of the k^{th} machine be:

  $$w_k = \frac{s_k}{\sum_{j=1}^{K} s_j},$$

  where s_j is the speed of the j^{th} machine.

Then according to (Kurve et al. 2011a) the cost function at the i^{th} LP is given by.

$$C_i(r_i, \mathbf{r}_{-i}) = \frac{b_i}{w_{r_i}} \sum_{\substack{j:r_j=r_i \\ j\neq i}} b_j + \frac{\mu}{2} \sum_{j:r_j\neq r_i} c_{ij}, \tag{3}$$

where $\mathbf{r}_{-i}$ denotes the vector of assignment of all the nodes in $\mathbf{r}$ except that of the i^{th} node and μ denotes the relative weight given to the inter-machine potential rollback-delay cost. This can vary across simulation platforms. For example, if the participating machines are remotely connected, then this cost will be higher than if locally connected. In the function above, the computational cost of a node intuitively depends on two factors: the existing load on the machine assigned to the i^{th} LP, *i.e.*,

$$\sum_{j:r_j=r_i, j\neq i} b_j,$$

and the computational load that the i^{th} LP will bring to the machine, *i.e.*, b_i. If the computational load generated by the node is zero ($b_i = 0$), then the computational part of the cost should be zero, thus motivate multiplication by b_i. This cost incentivizes the node to

choose a machine that has relatively less existing load, thus encouraging load balancing at the system level. For example, if $\mu = 0$ then a node i which is currently assigned to machine r_i would choose to relocate to machine r_i^* only if

$$\frac{1}{w_{r_i}} \sum_{j:r_j=r_i, j\neq i} b_j > \frac{1}{w_{r_i^*}} \sum_{j:r_j=r_i^*} b_j. \tag{4}$$

In this way the load balancing mechanism is implicitly manifested in the local cost function. The second term in the sum represents the weight of edges that connect the i^{th} node with nodes in other partitions. This term incentivizes the node to choose a partition with which it is well connected.

In (Kurve et al. 2011a), we also showed that this cost function allows for the existence of a Nash equilibrium which means that there is a fixed point in the strategy space of all the LPs. A strategy profile $\mathbf{r} = (r_1^*, r_2^*, ..., r_N^*)$ is a Nash equilibrium if and only if

$$C_i(r_i^*, \mathbf{r}_{-i}^*) \leq C_i(r_i, \mathbf{r}_{-i}^*) \quad \forall r_i \in \{1, 2, ..., K\}\ \forall i \in V. \tag{5}$$

Thus, at Nash equilibrium every node, say node i, will not be able to improve its cost by unilaterally changing its current processor r_i^*, *i.e.*, provided that the decisions of all other nodes are given by the assignment vector $\mathbf{r}_{-i}^*$.

Appendix C: an alternate cost framework

Let us consider an alternate local cost function for the graph partitioning problem.

$$\tilde{C}_i(r_i, \mathbf{r}_{-i}) = \frac{b_i^2}{w_{r_i}^2} + \frac{2b_i}{w_{r_i}^2} \sum_{\substack{j:r_j=r_i \\ j\neq i}} b_j - \frac{2b_i}{w_{r_i}} \sum_j b_j + \frac{\mu}{2} \sum_{j:r_j\neq r_i} c_{ij}. \tag{6}$$

Note that this cost function also has the property of machine-to-machine level overhead similar to the cost function described previously.

Next, we state a centralized graph partitioning problem that reasonably models the total simulation execution time by minimizing the load variance across machines as well as the potential inter-machine rollback-delay cost. Let $\underline{\mathbf{X}}$ be a $K \times |V|$ assignment matrix such that $x_{ki} = 1$ if node i belongs to machine k; otherwise $x_{ki} = 0$. We require $\sum_k x_{ki} = 1$ $\forall i = 1, 2, ..., |V|$, *i.e.*,

$$x_{ki} = 1 \quad \text{when } r_i = k. \tag{7}$$

Let w_k be the normalized capacity of the k^{th} machine so that $\sum_k w_k = 1$. Then, the centralized LP assignment problem is:

$$\min_{\underline{\mathbf{X}}} \quad \tilde{C}_0(\underline{\mathbf{X}}) = \sum_{k=0}^{K} \left(\frac{\sum_{j\in V} x_{kj} b_j}{w_k} - \sum_j b_j \right)^2 + \frac{\mu}{2} \sum_{i,j} c_{ij} x_{ki} (1 - x_{kj}) \tag{8}$$

$$\text{subject to } \sum_k x_{kj} = 1\ \forall\, j \text{ and } x_{kj} \in \{0, 1\}\ \forall\, k, j.$$

The above standard formulation of the graph partitioning problem (Van Den Bout and Thomas Miller 1990) is a quadratic integer programming problem the convexity of which depends on the network graph. In most cases it will not be convex. We can think of decomposing this problem into a set of K subproblems each of which is solved by a single partition. However, with the constraints $\sum_k x_{kj} = 1,\ \forall\, j \in V$, such a decomposition

is difficult to realize. So, instead we study the effect of sequential node-by-node transfer on (8). We can prove that for the local node cost function (6), Nash equilibria exist at the local optima (minima) of the centralized cost function (8). Thus, we can define a new cost function for each node as given by (6). Note that at each of the locally optimal points of (8), none of the nodes can improve their costs (6) given that the node assignments of all other nodes remain constant. Hence, the assignment vectors at these points are the Nash equilibria for this game. And since the node decisions always perform descent on (8), there is convergence guaranteed to one of the equilibrium points.

Endnote

[a]High volume inter-LP communication is not just overhead or delays, but also indicates the threat of rollback.

Competing interests
The authors would like to declare that there are no competing interests.

Authors' contributions
The authors of this paper contributed to this work in the following way: AK and GK were involved in the conceptualization of the agent based paradigm for PDES. AK and KK contributed to the coding and software implementation of the model in NetLogo. KK contributed to the method of modeling rollbacks and designing experiments. All authors read and approved the final manuscript.

Acknowledgements
The local incentive cost-based LP assignment method was presented at the International Workshop on Modeling Analysis and Control of Complex Network (CNET) 2011 at San Francisco. This work is supported in part by the NSF under grant number 0916179.

Author details
[1]EE Department, The Pennsylvania State University, University Park, PA 16802, USA. [2]CSE and EE Department, The Pennsylvania State University, University Park, PA 16802, USA.

Received: 20 November 2012 Accepted: 28 March 2013
Published: 26 April 2013

References

Agent-based modeling vs. equation-based modeling: A case study and users guide. In *Proceedings of the, First International Workshop on Multi-Agent Systems and Agent-Based Simulation*: Springer-Verlag; 1998:10–25.

Agrawal, VD, Chakradhar ST: **Performance analysis of synchronized iterative algorithms on multiprocessor systems.** *IEEE Trans, Parallel Distributed Syst* 1992, **3**(6):739–746.

Axelrod, R: **The dissemination of culture a model with local convergence and global polarization.** *J Confl Resolution* 1997, **41**(2):203–226.

Bagrodia, R, Meyer R, Takai M, Chen Y, Zeng X, Martin J, Song HY: **Parsec: a parallel simulation environment for complex systems.** *Computer* 1998, **31**(10):77–85.

Borshchev, A, Filippov A: **From system dynamics and discrete event to practical agent based modeling: reasons, techniques, tools.** In *Proceedings of the 22nd International Conference of the System Dynamics Society*; 2004. number 22.

Bu, T, Towsley D: **On distinguishing between internet power law topology generators.** In *Proceedings of the 21st, Annual Joint Conference of the IEEE Computer and Communications Societies IN-FOCOM*: volume 2. IEEE; 2002:638–647.

Carl, G, Kesidis G: **Large-scale testing of the Internet's Border Gateway Protocol (BGP) via topological scale-down.** *ACM Trans Model, Comput Simul (TOMACS)* 2008, **18**(3):1–30.

Chandy, KM, Misra J: **Asynchronous distributed simulation via a sequence of parallel computations.** *Commun ACM* 1981, **24**(4):198–206.

Chertov, R, Fahmy S: **Forwarding devices: From measurements to simulations.** *ACM Trans Model and, Comput Simul (TOMACS)* 2011, **21**(2):12.

Dimitropoulos, X, Krioukov D, Vahdat A, Riley G: **Graph annotations in modeling complex network topologies.** *ACM Trans Model, Comput Simul (TOMACS)* 2009, **19**(4):17.

Dorigo, M, Birattari M, Stutzle T: **Ant colony optimization.** *Comput Intell, Mag, IEEE* 2006, **1**(4):28–39.

Gupta, D, Vishwanath KV, Vahdat A: **Diecast: testing distributed systems with an accurate scale model.** In *Proc. 5th USENIX Symposium on Networked Systems Design and Implementation*: USENIX Association; 2008:407–422.

Jefferson, DR: **Virtual time.** *ACM Trans Program, Languages Syst (TOPLAS)* 1985, **7**(3):404–425.

Jennings, NR, Commun ACM: **An agent-based approach for building complex software systems.** 2001, **44**(4):35–41.

Karypis, G, Kumar V: **Parallel multilevel k-way partitioning scheme for irregular graphs.** In *Proc. 1996 ACM/IEEE Conference on Supercomputing*: IEEE; 1996.

Kurve, A, Griffin C, Kesidis G: **A graph partitioning game for distributed simulation of networks.** In *Proceedings of the 2011 International Workshop on, Modeling, Analysis, and Control of Complex Networks*: ITCP; 2011a:9–16.

Kurve, A, Griffin, C, Kesidis, G: **Iterative partitioning scheme for distributed simulation of dynamic networks.** In *Proc. 16th IEEE International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*: IEEE; 2011b:92–96.
Kurve, A, Griffin C, Miller DJ, Kesidis G: **Optimizing Cluster Formation in Super-Peer Networks via Local Incentive Design.**: Springer; 2013:Accepted.
Monderer, D, Shapley LS: **Potential games.** *Games Econ, Behav* 1996, **14:**124–143.
Niazi, M, Hussain A: **Agent-based computing from multi-agent systems to agent-based models a visual survey.** *Scientometrics* 2011, **89**(2):479–499.
Nicol, D, Fujimoto R: **Parallel simulation today.** *Ann Oper, Res* 1994, **53**(1):249–285.
Nikolai, C, Madey G: **Anatomy of a toolkit: A comprehensive compensium of various agent-based modeling.** In *Proceedings of the Agent 2007 Conference on Complex Interaction and Social Emergence*; 2007:87–92.
North, M, Howe T, Collier N, Vos J: **A declarative model assembly infrastructure for verification and validation.** In *Advancing Social, Simulation: The First World Congress*: Springer; 2007:129–140.
Pothen, A, Simon HD, Liou KPetal: **Partitioning sparse matrices with eigenvectors of graphs.** *SIAM J Matrix Anal Appl* 1990, **11**(3):430–452.
Psounis, K, Pan R, Prabhakar B, Wischik D: **The scaling hypothesis: Simplifying the prediction of network performance using scaled-down simulations.** *ACM SIGCOMM Comput Commun, Rev* 2003, **33**(1):35–40.
Reynolds, R: **Flocks, herds and schools: a distributed behavioral model.** In *ACM SIGGRAPH Comput Graph*: volume 21. ACM; 1987:25–34.
Sriram, K, Montgomery D, Borchert O, Kim O, Kuhn DR: **Study of BGP peering session attacks and their impacts on routing performance.** *IEEE J Selected Areas, Commun* 2006, **24**(10):1901–1915.
Tisue, S, Wilensky U: **Netlogo: A simple environment for modeling complexity.** In *Proc. International Conference on Complex Systems*: Citeseer; 2004:16–21.
Van Den Bout, DE, Thomas Miller III TK: **Graph partitioning using annealed neural networks.** *IEEE Trans Neural Netw* 1990, **1**(2):192–203.
Xu, J, Chung MJ: **Predicting the performance of synchronous discrete event simulation.** *IEEE Trans Parallel, Distributed Syst* 2004, **15**(12):1130–1137.
Zeigler, B, Praehofer H, Kim TG: **Theory of modeling and simulation: integrating discrete event and continuous complex dynamic systems.** *Academic Pr* 2000.
Zhang, L: **Internet topology data.** 2013. [http://irl.cs.ucla.edu/topology]

doi:10.1186/2194-3206-1-12
Cite this article as: Kurve *et al.*: An agent-based framework for performance modeling of an optimistic parallel discrete event simulator. *Complex Adaptive Systems Modeling* 2013 **1**:12.