Simulation of Scale-Free Networks

Gabriele D'Angelo
Department of Computer Science,
University of Bologna
Mura Anteo Zamboni 7, 40127
Bologna, Italy
gdangelo@cs.unibo.it

Stefano Ferretti
Department of Computer Science,
University of Bologna
Mura Anteo Zamboni 7, 40127
Bologna, Italy
sferrett@cs.unibo.it

ABSTRACT

In this paper, we present a new simulation tool for scale-free networks composed of a high number of nodes. The tool, based on discrete-event simulation, enables the definition of scale-free networks composed of heterogeneous nodes and complex application-level protocols. To satisfy the performance and scalability requirements, the simulator supports both sequential (i.e. monolithic) and parallel/distributed (i.e. PADS) approaches. Furthermore, appropriate mechanisms for the communication overhead-reduction are implemented. To demonstrate the efficiency of the tool, we experiment with gossip protocols on top of scale-free networks generated by our simulator. Results of the simulations demonstrate the feasibility of our approach. The proposed tool is able to generate and manage large scale-free networks composed of thousands of nodes interacting following real-world dissemination protocols.

Categories and Subject Descriptors

C.2.4 [**COMPUTER-COMMUNICATION NETWORKS**]: Distributed Systems—*Distributed applications*

General Terms

Algorithms, Experimentation, Performance

Keywords

Simulation, Scale-Free Networks, Parallel and Distributed Simulation, Performance Evaluation

1. INTRODUCTION

The fact that most real (bio/techno)-logical networks exhibit complex connectivity patterns is a well-known and widely accepted phenomenon [4, 33]. Whatever the role of nodes in the network under consideration (e.g. human being, computer, Web page), when described through graphs these complex networks share the distinctive feature of possessing nodes which tend to link each other following a degree distribution (i.e. number of neighbors) that can be well approximated by a power law distribution. Based on this, these networks are often referred to as *scale-free* networks. Edges in the graph can represent, for instance, sexual contacts among humans, connections among routers in the Internet, hypertextual links in Web pages, sensors connected in some ad-hoc manner. In any of these cases, hub nodes are present which possess high numbers of links with other nodes in the net, quite above the average node degree. The peculiar characteristics of these networks now drive more and more scientists to study their behavior. For instance, many mathematicians and physicists try to find analytical tools to represent them in the most accurate way [4, 33]. Biologists, sociologists and (again) physicists try to empirically demonstrate that more and more real nets possess scale-free features [27, 30].

As a matter of fact, a main problem is the lack of a powerful tool able to support the simulation of these nets. Real scale-free networks are usually composed of a huge amount of nodes. Thus, while mathematicians tend to prove their theorems in the limit of infinity (i.e. $n \to \infty$, n being the number of nodes in the network), commonly used simulation tools only allow to create relatively small scale-free networks [21, 41]. Under the simulation viewpoint, the modelling of large scale-free networks implementing complex communication protocols is a challenging task. The amount of memory necessary to model such networks is often huge such as the communication requirements among the nodes. The presence of hub nodes in a scale-free network, and in general the heterogeneity of nodes, is a key issue to be considered when designing simulators. In particular, imbalances in the computation and communication load can lead to very unsatisfactory results in case of parallel or distributed execution architectures.

We aim to offer a better characterization of the problem and to propose new methods to solve this scalability limitation. To demonstrate the validity of our approach, we present PaScaS (Parallel and distributed Scale-free Network Simulator), a novel simulator able to represent large scale-free networks, and manage them in a responsive way. The simulator provides a simple and fast method to build scale-free networks and to model information sharing and application protocols above them. It is worth noting that, following this approach, it is possible to define the characteristics of each single simulated node (e.g. CPU, memory, bandwidth and so on), modelling its local resources and pro-

SIMUTools '09, Rome, Italy

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5672
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5672

tocols. This aspect results very interesting in the simulation of real world scenarios. PaScaS is built on top of a simulation middleware (see Section 3.1) that provides support for the implementation of both sequential (i.e. monolithic) and parallel/distributed simulations.

In this work, we have at least two goals: i) propose a new tool for the efficient simulation of scale free networks, ii) investigate simulation approaches based on parallel and distributed execution architectures. In our opinion, following the latter approach, it should be possible to model very large networks and, in some cases, to speed up the simulation execution. Our aim is to provide a tool that promotes a new way to work with scale-free networks, allowing the network modelers to focus on dissemination protocols without facing the simulation details. Moreover, the scalability of the proposed tool should be able to permit the evaluation of the proposed protocols in more realistic environments.

To put evidence of our claim, we study how gossip protocols behave on top of scale-free networks. It is well-known that gossip schemes can easily spread information through networks [26]. We thus run three gossip algorithms on top of the scale-free networks we simulate through our tool. Outcomes of this study will be twofold. Firstly, we demonstrate that the simulator is able to easily manage networks of order of 10^4 nodes in a reasonable execution time (from 185 up to 3302 seconds, depending on the simulated gossip protocol). Secondly, in future works we will be able to evaluate the effectiveness of gossip protocols on top of scale-free networks. It is worth noting that, the proposed simulator is not specific to scale free networks, its components can be (and have been) used for the simulation of "generic" networks and scenarios. Focusing on the methods that have been implemented to enhance the simulator scalability, these are not specific to scale free networks, but scale-free networks, due to their characteristics, represent a new and very interesting testbed.

The remainder of this paper is organized as follows. Section 2 reviews the main principles at the basis of scale-free networks. Section 3 presents PaScaS, the Parallel and distributed Scale-free network Simulator. Section 4 describes the gossip algorithms we run on top on the generated networks. In Section 5 we report on an extensive simulation we performed to assess the performances of PaScaS, in presence of different execution environments and tunings of the simulator. Finally, in Section 6 we provide some final remarks.

2. SCALE-FREE NETWORKS

In this section we review the main principles at the basis of scale-free networks.

2.1 Background

Scale-free networks are gaining more and more attention in the research communities of computer science, physics, mathematics and biology. The reason is that these networks are quite good to model several types of real networks [4, 33]. Examples are the Web [4, 11], Internet [17], evolving networks in biology [27], citation graphs [35], social networks, sensor networks and so on [30].

Scale free networks are characterized by the fact that their nodes have a degree k (i.e., number of neighbor nodes attached to them) which is distributed according to a power law distribution, i.e., if p_k is the probability that a node has a degree equal to k, then $p_k \sim k^{-\alpha}$, for some constant α. (Sometimes, a cutoff is introduced to force that no node may have a degree higher than a given threshold value k_{max}.) It has been empirically calculated that usually in real networks $2 < \alpha < 3$.

Basically, the structure of such kind of networks is characterized by a notable (i.e., non negligible) presence of nodes, usually referred as *hubs*, which have a number of edges quite higher than the average degree in the network. Moreover, as in others kinds of networks, a major fraction of the set of nodes is connected (usually of the order of $\Theta(n)$, if n is the number of nodes in the network). Such main set of connected vertices is usually termed the *giant component*. Interestingly, it has been mathematically proven that when $2 < \alpha < 3$, the diameter of the network $d \sim \ln \ln n$, smaller even than small world networks, which remains almost constant while the network is growing [13]. It has been shown that these networks are quite resilient to random node faults, since the presence of hubs guarantees that the network remains connected. Indeed, the majority of nodes are those with small degree; thus, it is more likely that these ones will fail, while the probability that all hubs are eliminated is almost negligible. On the other hand, studies have shown that if one selects only hubs as the faulty nodes, the network rapidly becomes not connected, with several isolated graphs. This means that scale-free networks are not tolerant to targeted attacks to nodes with higher degrees [1, 11, 33]. This is an important result, for instance, to study and prevent security attacks in computer networks such as denial of service.

2.2 Building Scale-Free Nets

A practical method to build a scale-free network was introduced by Barabási and Albert in [3] and works as follows. The scheme proceeds in discrete time steps. It starts with a default number of nodes n_0 in the network. At each time step, a new node is added to the network, with initial degree m. For each edge of the new node, a neighbor is selected and a link between the two nodes is created. The neighbor node is selected with a probability proportional to the degree of that vertex, i.e. the higher the degree of a node the more likely it will be selected as a neighbor of the newly added node. This approach is often referred to as *preferential attachment*, and perfectly models the *rich get richer* phenomenon, arising when the amount an entity gets in time, goes up with the amount it already has [33]. Indeed, the preferential attachment is responsible for the generation of a power law node degree distribution. As a consequence of this scheme to generate a scale-free network, the average degree of a given node $\langle k \rangle \simeq 2m$, for large networks. Indeed, due to the fact that each new node is added at each time step, with new m links, at time t the network has $n_0 + t$ nodes, with mt links. n_0 can be considered as a negligible term for large nets (or large times), hence confirming this claim, i.e. $\langle k \rangle \simeq 2m$ (each link counts for two node edges).

2.3 Notation

We model the network being built and simulated as a set of distributed nodes. The topology of the network is defined as a graph $G = (\Pi, L)$, where $\Pi = \{n_1, n_2, \ldots\}, |\Pi| = n$, is the set of nodes,[1] and L denotes the set of edges among nodes in the network. Two nodes n_i, n_j are neighbors if an

[1]Hereinafter we will use the term *node* to refer to the vertex in the network being simulated, trying to avoid any confu-

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5672
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5672

edge $l_{ij} \in L$ exists connecting the two nodes in G. The set of neighbors of n_i is denoted with Π_i.

3. A SCALE-FREE NET SIMULATOR

The simulation of networks is a very wide field of research that, in the last years has produced many valuable results. Almost all aspects of networking have been deeply investigated, and many general and specific tools have been proposed [37, 24, 36].

Despite this extensive effort, there are few works in the literature on specific simulators of scale-free networks. For instance, [28] shows how to build a scale-free network to simulate air transportation networks. Focus is given on the scheme to build the network, rather than the need to have an effective simulation tool itself. Instead, [15] proposes a model to simulate scale-free networks; they start from a simulator running on a single-CPU, and compare it with a distributed environment with parallel clustered processors.

PaScaS (Parallel and distributed Scale-free Network Simulator) is a new tool specifically designed for the modelling of scale-free networks. Our goal is to provide a scalable and easy-to-use simulator that can be used to design new application-level protocols and to evaluate the performance of existing. Each PaScaS simulation starts with the building of a scale-free network of a given size. In the current version, to build the scale-free network, PaScaS runs the algorithm proposed by Barabási and Albert in [3]. We plan to add a more comprehensive set of building methods in the following releases. After completing the network topology, each node interacts with the others depending on its local characteristics and the implemented network protocol. It is worth noting that, in this way, it is possible to test environments that are very heterogeneous in terms of node characteristics and behaviors. For example it would be possible to evaluate the performance of gossip protocols that are defined in a different way if running on a leaf network node or a hub.

PaScaS can be used to implement both sequential and parallel/distributed simulations, depending on the scalability and performance requirements. The core of the simulator is the Advanced RTI System (ARTÌS) middleware (Section 3.1). Focusing on Parallel And Distributed Simulation (PADS), the performance of the simulator can be enhanced thanks to the GAIA framework 3.2, that provides functionalities to reduce the communication overhead in parallel and distributed simulation.

3.1 The Advanced RTI System (ARTÌS)

The Advanced RTI System (ARTÌS) is a middleware specifically designed for parallel and distributed simulation, but that can be used also to build efficient monolithic simulations [2]. Its design, partially inspired by the High Level Architecture (HLA, IEEE 1516) [23], is specifically tailored to handle with high degree of model scalability and execution architectures composed of large execution clusters.

The middleware has been used as a testbed for the design and development of new features such as: simulation cloning and concurrent replication, communication marshalling, and the study of specifically tailored data structures for the management of simulation events. Some new features have been introduced to improve scalability and simulator performance, and a simplified set of Application Programming Interfaces (APIs) has been provided to facilitate the development of PADS. As well known, synchronization is a main issue in PADS, therefore the simulation middleware has to provide time management services to the simulation components composing the distributed simulation. For the sake of generality, ARTÌS supports both conservative (Chandy-Misra-Bryant, time-stepped) [32, 18] and optimistic (Time Warp) [25] synchronization algorithms. The current version of PaScaS follows a synchronization scheme based on time-steps. The middleware is freely available for research purposes and can be downloaded from [2].

sion with the logical processes where the distributed simulation actually runs.

3.2 Generic Adaptive Interaction Architecture (GAIA)

One of the main bottlenecks of parallel and distributed simulation is the communication cost due to interactions among simulated entities. In a monolithic (sequential) simulator, the simulated entities easily interact accessing the memory allocated from the simulator software. In a parallel or distributed environment composed of many execution units, this low-cost approach is not possible; the alternatives are: shared-memory when available (e.g. multi-processors) or LAN / Internet -based communication in the other cases. Due to this factor, the communication cost results as one of the main factors affecting the performance of parallel and distributed simulation. It follows that the reduction of the communication cost and the load balancing are main fields of PADS research. Focusing on the partitioning of simulation models, many different approaches have been proposed to maintain a global shared-state in distributed simulation, while dynamically filtering the event- and state-information to reduce the communication overhead. Some examples are static partitioning schemes [16], spheres of influence [31], simulation domains [40], data distribution management [23], dynamically adaptive partitionions [29] and hierarchical federations [12]. Despite many works have addressed the problem of load-balancing in parallel simulation environments [39, 14, 9, 20, 38], only a few have examined the problem of simulation model partitioning considering both communication cost and load-balancing requirements in parallel distributed simulation environments. For example, in [34] is proposed a dynamic partitioning algorithm for optimistic distributed simulation. In our vision, the reduction of the communication cost and the aspects of load-balancing in the parallel or distributed architecture can be seen as different aspects of the same problem and therefore a joint approach is necessary [19].

The Generic Adaptive Interaction Architecture (GAIA) [8] is a migration based framework that uses the services provided by the ARTÌS middleware. Following the GAIA paradigm, each entity in the simulation can be migrated within the execution architecture, with the aim to cluster (migrate) the highly interacting entities in the same execution units, and therefore reduce the communication overhead (see Section 3.2.1). This version of GAIA is a very simplified form of the mechanism presented in [19]. In brief, the proposed mechanism continuously audits the communication pattern of each simulated entity, and determines if a better allocation is possible. In the past we have demonstrated that this approach leads to valuable results in the simulation of wireless devices and cooperative multi-agent

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5672
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5672

systems. In this work, we demonstrate that it can be extended also to deeply different scenarios such as scale-free networks. It is worth noting that, the mechanism could be further adapted and tuned to this scenario, conversely we are interested to verify if a very simple form of the mechanism is able to provide an advantage with respect to traditional approaches (e.g. without any form of entity migration) to parallel and distributed simulation.

Under the development viewpoint, the framework provides to the simulation developer an easy-to-use entity based paradigm for the definition of models. Following this approach, the level of abstraction provided to the developer is relatively high. In this sense, the ARTÌS middleware is completely transparent to the developer. GAIA provides to the simulation model the communication services and the support for entity migration.

3.2.1 The Clustering Mechanism

Following the GAIA approach, a simulation can be decomposed in a set of interacting Simulated Model Entities (SME), in which each SME models the evolution of a part of the system and interacts with other SMEs following a message-based approach. Given the not negligible cost of communication in parallel and distributed execution environments, one goal of the mechanism is to cluster the highly interacting SMEs in the same execution unit. In this way, it could be possible to reduce the communication cost.

Due to the highly dynamic and unpredictable nature of the models of interest, we propose to adaptively reallocate (i.e. migrate) the SMEs over the available execution units. This can be implemented auditing the communication pattern of each SME during the simulation execution and evaluating if reallocations are necessary. In practical terms, the migration can be implemented as data structures transfer (i.e. the internal state of SMEs) but it is worth noting that the cost the migrations is not negligible and has to be carefully considered. In GAIA, these aspects are implemented using two main heuristics: i) the "base heuristic" and ii) the "group heuristic", that are evaluated at runtime.

The *base heuristic* analyzes the communication pattern of each SME and determines if it should be migrated from the current execution unit to another. To reduce the overhead of the mechanism, at the end of each synchronization phase, only the SMEs that have sent at least one interaction are taken in account. If a SME has much of its interactions delivered outside the "local" execution unit then it is a candidate for migration, and the destination is the execution unit that receives the larger percentage of outbound interactions. To avoid unnecessary migrations, this is actually done only if the chosen destination receives more messages than the local execution unit. This general scheme is refined and tuned using specifically defined parameters that determine aspects such as: the size of imbalance needed to trigger a migration and the amount of time between two migrations of the same SME. Furthermore, to introduce a weighting method, a sliding-windows scheme is introduced in the mechanism. In detail, the interactions have a different weight that depends on its aging and payload size. This is necessary because the heuristic needs to have up-to-date data as input, elsewere its performance would be severely degraded. For example, interactions that are too old should not be included in the data set: they are related to a simulated model state that could be very different from the current one. At the same time, focusing only on the interactions delivered in the current state, drastically reduces the possibility to perform trend analysis and could lead to underreaction or overreaction in the adaptive mechanism. Finally, interactions with different payload size should be treated differently: in fact a correction factor that is proportional to the size (with respect to the average) is applied.

The results of the basic heuristic are analyzed and modified by the *group heuristic*. In this case, the goal is to evaluate groups of SMEs instead of single entities. In fact, single SMEs that are not candidate could be migrated due to the migration of all SMEs that are interacting with them. In these terms, the group heuristic tries to achieve a higher level of abstraction with respect to the base one. First of all, the implementation of the group heuristic tries to find the groups of interacting SMEs. For each group that is found, the heuristic analysis the communication pattern of each SME that is in the group. In this way, it determines if other SMEs (that are local to the execution unit) should be migrated, aiming to maintain the clustering of the whole group in the same execution unit.

The clustering mechanism described above is constrained by the computational load-balancing requirements of the parallel or distributed execution architecture. In other words, the clustering of SMEs has to be done taking care of the load of each execution unit. It is obvious that an unconstrained clustering mechanism would cluster all the SMEs in the same execution unit, that is the worst case for load balancing. Complex load-balancing schemes have been proposed and implemented in GAIA [19], while in the actual version of PaScaS a quite simple mechanism, that maintains a constant number of SMEs in each execution unit [5], has been used with the aim to reduce at the bare minimum the overhead introduced by the mechanism.

It is worth noting that, the approach introduced above is not specific to scale free networks and in the past it has been applied to many other models (e.g. wireless networks) with interesting results. In these terms, for the reasons described in the first part of this paper, the scale free networks represent a new challenging environment to verify its applicability. As only an overall description of the mechanism has been given, due to space limitations, more details can be found in [8, 5, 6] and [19].

3.3 PaScaS

The core of PaScaS is a simulation model implemented using the APIs provided by GAIA, for performance reasons this version of PaScaS is written in C language. The model implements the main features of the scale free network simulator such as the building algorithms, the behavior and the characteristics of each node and the gossiping protocols. The set up of the simulated scenarios, and the tuning of the runtime parameters of the simulator, is obtained via configuration files and environment variables. A set of scripts is provided to facilitate and automatize the execution of parallel and distributed runs. This approach has been chosen to facilitate the set up of unattended batch executions. The results of the runs are collected in logging files, tuned to the adequate level of detail that has been chosen by the simulation modeler. Other scripts are available to collect and analyze the requested data and results. Furthermore, as said in Section 1, a main aspect of PaScaS is the possibility to build heterogeneous scenarios, that is models in

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5672
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5672

which each simulated model has specific characteristics or configurations with respect to other nodes. For example, different gossiping protocols (i.e. hubs vs. leaf nodes) and simulated hardware characteristics (es. available memory). This aspect is made available to the end user properly defining the configuration files. The simulator has demonstrated to be very simple and easy to use, but to further improve its usability, we are currently developing a graphical user interface that can be used for the configuration, automatically generating the configuration files, setting the environment variables and launching the batch scripts.

PaScaS is freely available for educational and research purposes and is going to be part of ARTÌS version 2 that is to be released shortly [2].

4. GOSSIPING PROTOCOLS

In this section, we describe the protocols we consider to gossip messages on scale-free networks, as an example of use of our simulator.

According to our model, all nodes are able to generate a new message to be disseminated in the network. When the generation procedure is invoked at a given node, a single message may be created with a certain probability, as described in Algorithm 1. The generation of a message simulates the occurrence of a new event produced at a given node that must be propagated. If the message is created, then it is gossiped through the net, using a GOSSIP() procedure (line 6 of the algorithm). The message is also inserted in a cache (line 5).

Algorithm 1 Generation of a Message

```
1: function GENERATE()
2: t ← GENERATIONTHRESHOLD()
3: if RANDOM() < t then
4:    msg ← CREATEMESSAGE()
5:    CACHE(msg)
6:    GOSSIP(msg)
7: end if
```

Algorithm 2 Reception of a Message

```
1: function RECEIVE(msg)
2: if (NOTCACHED(msg) ∧ msg.ttl > 0) then
3:    CACHE(msg)
4:    msg.ttl ← msg.ttl − 1
5:    GOSSIP(msg)
6: end if
```

Upon reception of a given message (see Algorithm 2), the receiving node forwards the message to its neighbors using the gossiping protocol by calling the GOSSIP() function (line 5 in the algorithm). This is accomplished only if the message is not already in the node's cache. The idea is that if the message is in cache, it has already been gossiped; hence, the node has nothing to do with the message *msg* (line 2). Conversely, *msg* is gossiped and cached (line 3 of Algorithm 2). Needless to say, due to the possible memory constraints of a node, the cache is limited in size (*cache.size*). Think for instance, at the simulation of some kind of scale-free, ad-hoc networks [10, 21]. Hence, upon insertion of a message in the cache, a control is performed on the cache; if it is full, an old message is removed. We do not provide here the complete description of the CACHE() procedure; we simply implemented an aging policy to free memory in cache. Moreover, in order to avoid that old messages are indefinitely propagated among nodes, due to the limited size of the cache which cannot contain the complete list of messages forwarded in the past, a time-to-live parameter is inserted within each message (i.e. *msg.ttl* in the algorithm), which is progressively decreased (line 4) until it reaches a 0 value; in this case the message is not forwarded (see the second part of the condition in line 2).

We consider three different algorithms which implement the GOSSIP() procedure. These are shown in Algorithms 3, 4 and 5. All algorithms require the definition and initialization of a parameter at each node, defined through the INITIALIZATION() procedure reported at the beginning of these algorithms.

4.1 Gossip #1: Fixed Probability of Dissemination

According to the first gossip protocol, Algorithm 3, the node (say n_i) randomly selects those edges through which the message *msg* must be propagated [21, 41]. Specifically, all n_i's neighbors (i.e. Π_i) are considered and a threshold value $v \leq 1$ is maintained, which determines the probability that *msg* is gossiped to the neighbor (when $v = 1$ we obtain a flooding algorithm). At each step the message is propagated from n_i to $v|\Pi_i|$ other nodes. (In a scale-free network, on average a given node will propagate the message to $v\langle k\rangle \simeq 2vm$ nodes.) Hubs will send a higher number of messages to their neighbors, with respect to others. This is in perfect accordance with the nature of scale free networks, since each node contributes to disseminate the message in accordance with its degree. This also means that the work (in terms of computation and communication) performed at hubs is higher than at other nodes.

Algorithm 3 Gossip: Fixed Prob. of Dissemination (at n_i)

```
1: function INITIALIZATION()
2: v ← CHOOSEPROBABILITY()
3:
4: function GOSSIP(msg)
5: for all n_j ∈ Π_i do
6:    if RANDOM() < v then
7:       SEND(msg,n_j)
8:    end if
9: end for
```

4.2 Gossip #2: Fixed Fanout

We contrast the gossip scheme above against another approach, reported in Algorithm 4. According to it, a message is sent to a fixed number of nodes (i.e. a fixed fanout is exploited), selected at random among the n_i's neighbors, Π_i [21]. This means that the higher the degree of n_i, the more unlikely a n_i's neighbor will receive a gossip message at each step. (When $fanout = 1$, the scheme resembles a search scheme where at each step a single neighbor is randomly contacted). In this case, a constant *fanout* is chosen and shared among all nodes in the network, during the INITIALIZATION() procedure. When a message *msg* is to be gossiped, n_i selects a number of neighbors equal to the *fanout*. A list of nodes (*toSend* in the algorithm) is filled up by iteratively selecting a node among the neighbors not already in the list (see lines

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5672
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5672

8-12).[2] If the number of neighbors is lower than the selected fanout, the message is sent to all the n_i's neighbors, see lines 5-6).

Algorithm 4 Gossip: Fixed Fanout (at n_i)

```
1: function INITIALIZATION()
2: fanout ← RETRIEVESHAREDFANOUT()
3:
4: function GOSSIP(msg)
5: if fanout ≥|Π_i| then
6:    toSend ← Π_i
7: else
8:    toSend ← ∅
9:    for i = 1 to fanout do
10:      select n_j ∈ Π_i ∩ toSend, i ≠ j
11:      toSend ← toSend ∪ n_j
12:   end for
13: end if
14: for all n_j ∈ toSend do
15:   SEND(msg,n_j)
16: end for
```

4.3 Gossip #3: Probabilistic Broadcast

The third distribution protocol we consider is a probabilistic broadcast scheme (see in Algorithm 5). Once the GOSSIP() procedure is called, if the message has been locally generated at the node and *msg* still needs to be spread to the network (we assume this check is performed in FIRSTTRANSMISSION(), line 5), *msg* is sent to all node's neighbors (lines 6-8). Conversely, if *msg* has been received from someone else, the node decides with a certain probability p_b (defined at the beginning of the protocol) to forward *msg* (line 5). In the positive case, the message is sent to all node's neighbors.

Algorithm 5 Probabilistic Broadcast

```
1: function INITIALIZATION()
2: p_b ← PROBABILITYBROADCAST()
3:
4: function GOSSIP(msg)
5: if (RANDOM() < p_b ∨ FIRSTTRANSMISSION()) then
6:    for all n_j ∈ Π_i do
7:       SEND(msg,n_j)
8:    end for
9: end if
```

Such algorithm can be exploited to simulate message propagation in MANETS, VANETs, or sensor networks, where the transmission of a message corresponds to a broadcast of a message, received by all nodes into the wireless network coverage of the message sender. In this case, no overlay is exploited and all receiving nodes process *msg*. Then, they decide to retransmit it with a certain probability, in order to limit the flooding of the message and avoid congestion of the wireless network.

5. EXPERIMENTAL EVALUATION

In this section it will be investigated the performace of PaScaS in different configurations and scenarios. We will start considering the monolithic version of the simulator. Then, these results will be compared to a parallel version of PaScaS running on a set of execution units. Finally, in the last part of this section we will consider the effect of the proposed adaptive mechanism (Section 3.2) on the performances of a parallel simulation execution.

Considering the scalability of the proposed tool, it is worth noting that, due to their nature, scale-free networks have some characteristics that should be carefully taken in account. In detail, doubling the number of nodes composing a scale-free network, more than doubles the number of links in the network. When implementing gossip protocols or in general dissemination protocols, this aspect becomes of fundamental importance. For example, doubling the number of nodes in a simulated model, will at least double the amount of computation required to manage the nodes, but the growth in the amount of communication required by the dissemination protocol will be much higher. In terms of simulation, this translates to models with an increasing communication requirements with respect to computation needs. In the case of parallel and distributed simulation, that is highly affected by the amount of communication in the simulated model, this aspect can lead to severe limitations in scalability.

5.1 Sequential Execution

The experimental evaluation starts with the scalability evaluation of PaScaS in a sequential (monolithic) configuration and comparing the results when running the different gossip protocols introduced in Section 4.

Table 1: Model parameters and simulation scenario

Parameter	Value
Number of nodes	3000, 6000, 9000, 12000
m	2
Message generation	Exponential distribution Mean=50 timesteps
cache.size	10
msg.ttl	6 (fixed prob. & fanout) 4 (conditional broadcast)
Prob. of dissemination (v)	0.5 (i.e. 50%)
Fanout value	5
Prob. of broadcast p_b	0.5 (i.e. 50%)
Simulated time	1000 timesteps (after network building)

Table 1 reports the main simulation parameters and configuration values used in the performance evaluation. The results shown in Figure 1 are obtained running the monolithic simulator on a dual processor execution unit equipped with single-core Intel Xeon "Gallatin" CPU 2.80 GHz Hyper-Threading [22] processors (L2-Cache: 512 Kb, L3-Cache: 1024 Kb, Front Side Bus: 533 MT/s) and with 2 GB of RAM. In this case, the monolithic simulator is able to use only a single processor.

Figure 1 shows the rather good scalability of PaScaS: networks composed of up to 12000 nodes can be simulated with an acceptable execution time. It is quite evident that the implemented gossip protocols have different behaviors and requirements. The implementation of fixed probability (Gossip #1) requires the computation of many random numbers (i.e. one for each neighbor of a given node). This costly operation is necessary, for each message, to determine which

[2] $\overline{toSend}$ represents the complement of the set *toSend*.

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5672
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5672

neighbors should receive the message. Focusing on the implementation of the conditional broadcast (Gossip #3), the amount of computation required by the algorithm is more limited but, in this case, the execution time is mainly affected by communication. Due to its nature, the conditional broadcast, produces a very large amount of messages to be disseminated in the simulated network. Finally, the fixed fanout (Gossip #2), shows the best performances. This is due its very limited requirements, both in terms of computation and communication. In this case, the computation requirements are very low because the dissemination protocol selects the destination nodes using an heuristic mechanism that is based on the extraction of a single random number.

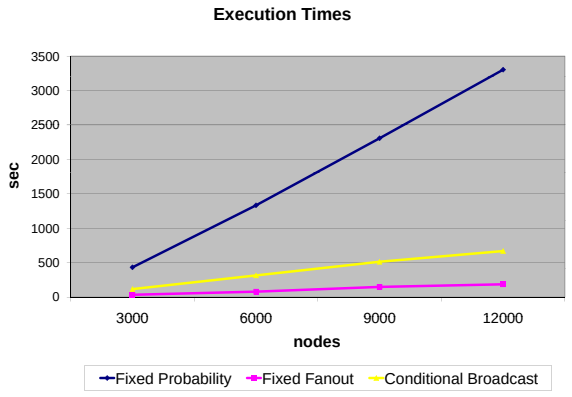

Figure 1: Execution Times. For Fixed Probability and Fixed Fanout $ttl = 6$; for Conditional Broadcast $ttl = 4$

5.2 Parallel Execution

In parallel and distributed simulation, the representation and evolution of the simulation model is obtained through the coordinated execution of a set of components (referred as Logical Processes, LPs). Each LP, manages the evolution of a part of the simulated model entities (e.g. network nodes), and is usually allocated on a different CPU. To obtain a correct execution of the simulation, the set of LPs have to be synchronized. This aspect has the effect to increase the communication requirements of the parallel/distributed execution with respect to the monolithic approach. Furthermore, as introduced in Section 3.2, the communication cost is strictly related to the execution platform (e.g. process memory vs. shared memory vs. Internet). For these reasons, the performance of a parallel/distributed simulator can be seen as a trade-off between the gain obtained thanks to the parallelization of computation (i.e. additional computational power) and the cost of communication introduced by the parallel/distributed environment.

In PaScaS, the allocation of simulated nodes among the LPs is randomized, that is for each node a random allocation LP is chosen. It is not possible to perform any "optimized" *a priori* allocation of nodes given that the scale-free network, in this phase, is still not built. It will be generated by a partially randomized building process. Furthermore, the implemented dissemination protocols show not predictable communication behaviors and patterns. Due to these reasons, any form of static pre-allocation of nodes among LPs would become suboptimal in a few timesteps and would not lead to any performance gain.

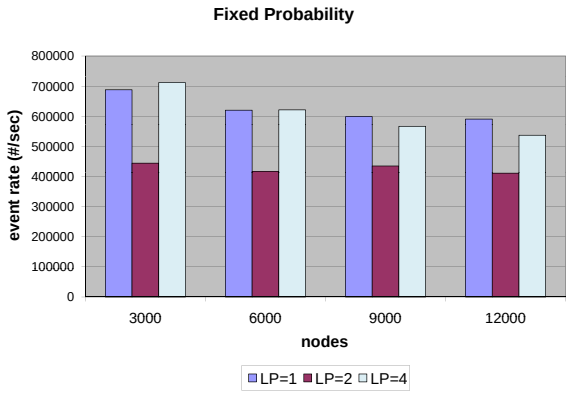

Figure 2: Fixed Probability

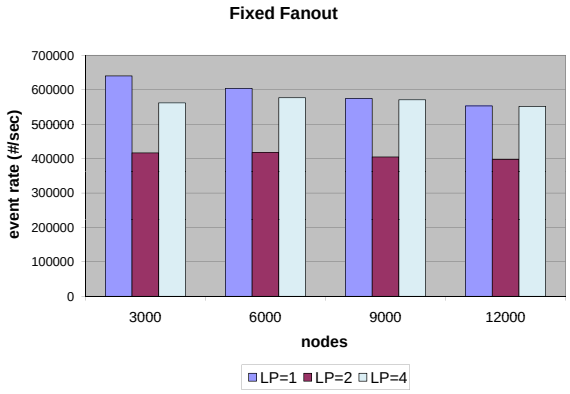

Figure 3: Fixed Fanout

Figures 2, 3 and 4 report the performance of PaScaS in a parallel simulation environment composed of an increasing number of LPs (i.e. 1, 2 and 4) and implementing the previously described gossip protocols. In this case, the measure of performance used to evaluate the simulator is the event rate, that is the total number of events processed per second. For better readability, we have summarized the results in Table 2. The Table reports the variation of the event rate when the simulations were run on a single LP (that is monolithic execution) and in parallel ($LP = 4$). In all cases, the performance obtained from a parallel execution with 2 LPs is lower with respect to a monolithic run. Therefore, the amount of parallelization provided by two CPUs is not enough to balance the communication overhead introduced by the parallel execution. Increasing the number of LPs ($LP = 4$), it is possible to achieve better performance and to almost match the results of the monolithic execution. This result, in a dual processor execution unit, it is not surprising and further confirms the results reported in [7]. In brief, the Hyper-Threading technology [22] makes each single physical

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5672
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5672

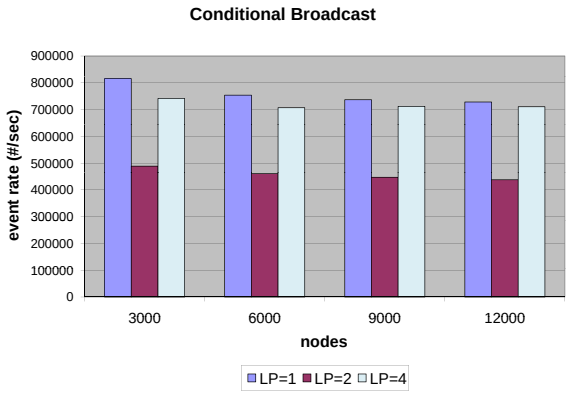

Figure 4: Conditional Broadcast

processor appearing as two logical processors[3] at the user's level. The most clear effect is that, in this case, the best performances are obtained when the number of LPs, in the simulation, is equal to the number of logical processors in the execution architecture.

Table 2: Performance gap (%) between $LP = 1$ and $LP = 4$; Gossip #1: Fixed Probability; Gossip #2: Fixed Fanout; Gossip #3: Conditional Broadcast

Nodes	Gossip #1	Gossip #2	Gossip #3
3000	3.46	-12.22	-9.1
6000	0.19	-4.49	-6.23
9000	-5.35	-0.63	-3.36
12000	-9.07	-0.25	-2.42

The results obtained through parallel execution are quite unsatisfactory, only in two cases there is a very limited gain. In all other cases the parallel version of the simulator is slower than monolithic. It is a clear demonstration that, in this case, the communication overhead among LPs introduced by the parallel execution not balances the gain given by the load parallelization. In detail, increasing the number of simulated nodes, the fixed probability dissemination (Gossip #1) shows a performance worsening of parallel with respect to monolithic (see Table 2). In this case, increasing the number of nodes, makes communication overwhelming computation, and therefore a slowdown of parallel execution with respect to monolithic. In the other cases (i.e. fixed fanout and conditional broadcast, Gossip #2 and #3, respectively), the ratio between computation and communication is highly unbalanced. In both cases, the communication requirements introduced by the simulation of a new node, is less with respect to its computational needs. This explains the better performances obtained by the parallel execution when increasing the number of nodes.

5.3 Adaptive Parallel Execution

As seen in the previous section, the communication overhead due to the parallel execution of the simulator has a very negative impact on performances. Our goal is to demonstrate that, an adaptive mechanism that reallocates simulated entities, can lead to a speed-up of parallel simulations. As described in Section 3.2, the GAIA framework analyzes step-by-step the communication pattern of each simulated entity (i.e. network nodes) and re-allocates them clustering the highly interacting ones in the same LP. In this way, it is possible to reduce the communication overhead due to the parallel nature of the simulation execution architecture.

Figures 5, 6 and 7 report the performance of PaScaS in an adaptive parallel simulation environment in different configurations (i.e. 1, 4 and 4 GAIA on) and implementing the previously described gossip protocols. In all cases, the adaptive approach leads to a performance increase with respect to a non-adaptive parallel (i.e. $LP = 4$) and monolithic (i.e. $LP = 1$) execution. In detail, in Table 3 are reported the performance gap between $LP = 1$ and $LP = 4$ *GAIA on*. In general the effect of the adaptive mechanism on performance is positive, with peaks higher than 30%. The best results are obtained for Gossip #1 and #3 which are the protocols with the higher amount of communication, that is where the adaptive mechanism is able to obtain a performance increase. Further increasing the number of simulated nodes, leads to a small decrease in performances, this is due again to the balance between communication and computation. When the communication requirements of the parallel architecture overwhelm the computation load, the adaptive mechanism is unable to compensate the loss. In this case, the gain obtained by the adaptive mechanism can only slowdown the decrease of performance.

Table 3: Performance gap (%) between $LP = 1$ and $LP = 4$ *GAIA on*; Gossip #1: Fixed Probability; Gossip #2: Fixed Fanout; Gossip #3: Conditional Broadcast

Nodes	Gossip #1	Gossip #2	Gossip #3
3000	34.18	-1.33	19.75
6000	38.63	6.87	23.37
9000	30.97	11.07	24.59
12000	26.47	9.65	22.56

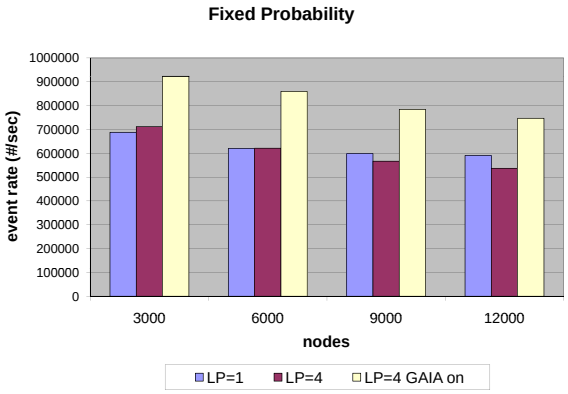

Figure 5: Fixed Probability with GAIA

[3]The logical processors should not be confused with logical processes (LPs).

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5672
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5672

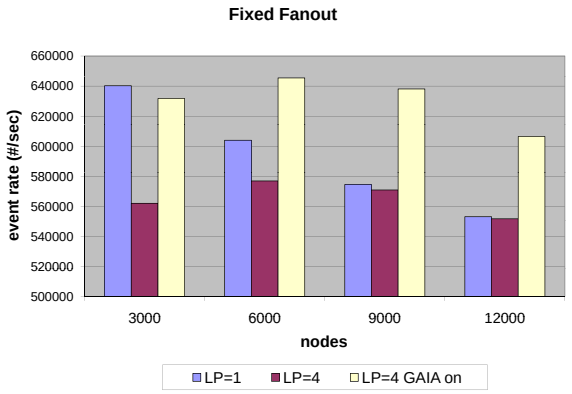

Figure 6: Fixed Fanout with GAIA

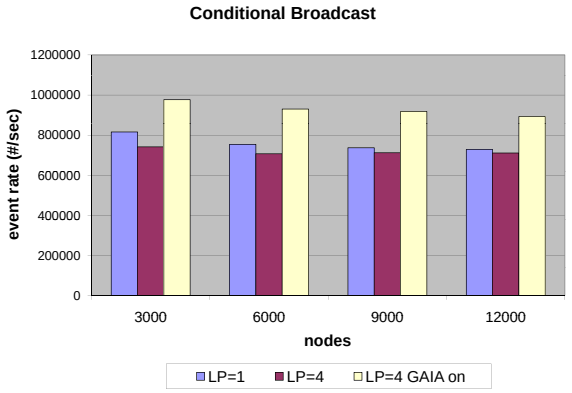

Figure 7: Conditional Broadcast with GAIA

Finally, we have tested the scalability of the proposed mechanism increasing the number of processors, and consequently of LPs. The results shown in Table 4 are obtained running the parallel simulator on a quad processor execution unit equipped with single-core Intel Xeon MP "Foster MP" CPU 1.50 GHz Hyper-Threading processors (L2-Cache: 256 Kb, L3-Cache: 1024 Kb, Front Side Bus: 400 MT/s) and with 2 GB of RAM.

Table 4: Performance gap (%) between $LP = 8$ and $LP = 8$ *GAIA on*; Gossip #1: Fixed Probability

Nodes	Gossip #1
8000	38.18
16000	45.37

The results show a very significant gain; in presence of an increasing number of LPs the adaptive mechanism is able to effectively reduce the communication overhead of the parallel execution. This behavior can be explained considering the probability that two simulated entities are allocated in the same LP. This probability decreases while increasing the number of LPs that compose the simulation. For this reason, a simulation composed by many LP has the effect to enhance the performance that can be obtained by the adaptive clustering mechanism.

6. CONCLUSIONS AND FUTURE WORK

We presented a novel scale-free network simulator, named PaScaS. The tool is able to create and manage complex networks with scale-free characteristics. The scalability of the tool permits the simulation of networks composed by a high number of nodes. PaScaS allows the implementation of both monolithic and parallel/distributed simulations. The results demonstrate that non-adaptive parallel simulations achieve unsatisfactory results due to the nature of scale-free networks and the overhead introduced by the parallel execution architecture. Conversely, the implementation of a mechanism based on adaptive migration of simulated entities, can lead to a valuable increase in performances, extending the scalability of the proposed tool.

As a future work, we plan to extend and tune our tool to many other specific execution architectures such as distributed environments and multi-core processors. In particular, we plan to design and implement more sophisticated heuristics for the adaptive re-allocation mechanism. On the other hand, we plan to extend our scale-free network model with more features and to focus on the performance evaluation of gossip protocols, to assess their effectiveness on dissemination information over large scale-free networks.

7. ACKNOWLEDGMENTS

The authors wish to thank Filippo Giunchedi for his contributions to the PaScaS simulator, and the anonymous reviewers for their very useful comments.

8. REFERENCES

[1] R. Albert, H. Jeong, and A.-L. Barabási. Error and attack tolerance of complex networks. *Nature*, 406, July 2000.

[2] ARTÌS: Advanced RTI System Homepage. http://pads.cs.unibo.it, 2009.

[3] A.-L. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, 286, 1999.

[4] A.-L. Barabási, R. Albert, and H. Jeong. Scale-free characteristics of random networks: the topology of the world-wide web. *Physica A: Statistical Mechanics and its Applications*, 281(1-4), Jun 2000.

[5] L. Bononi, M. Bracuto, G. D'Angelo, and L. Donatiello. A new adaptive middleware for parallel and distributed simulation of dynamically interacting systems. In *DS-RT '04: Proc. of the 8th IEEE International Symposium on Distributed Simulation and Real-Time Applications*. IEEE, 2004.

[6] L. Bononi, M. Bracuto, G. D'Angelo, and L. Donatiello. Performance analysis of a parallel and distributed simulation framework for large scale wireless systems. In *MSWiM '04: Proc. of the 7th ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems*. ACM, 2004.

[7] L. Bononi, M. Bracuto, G. D'Angelo, and L. Donatiello. Exploring the effects of Hyper-Threading on parallel simulation. In *DS-RT 06:*

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5672
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5672

Distributed Simulation and Real-Time Applications, IEEE International Symposium on. IEEE, 2006.

[8] L. Bononi, G. D'Angelo, and L. Donatiello. HLA-based adaptive distributed simulation of wireless mobile systems. In *Proc. 17th ACM/IEEE/SCS Workshop on Parallel and Distributed Simulation*. IEEE Press, 2003.

[9] A. Boukerche and S. Das. Dynamic load balancing strategies for conservative parallel simulations. In *PADS '97: Proc. of the eleventh workshop on Parallel and distributed simulation*. IEEE, 1997.

[10] A. Brayner and R. Menezes. Balancing energy consumption and memory usage in sensor data processing. In *SAC '07: Proc. of the 2007 ACM symposium on Applied computing*, New York, NY, USA, 2007. ACM.

[11] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener. Graph structure in the web. *Computer Networks*, 33(1), June 2000.

[12] W. Cai, S. Turner, and B. Gan. Hierarchical federations: an architecture for information hiding. *Parallel and Distributed Simulation, 2001. Proceedings. 15th Workship on*, 2001.

[13] R. Cohen, S. Havlin, and D. Ben-avraham. Structural properties of scale-free networks. In *In Handbook of Graphs and Networks*. Wiley, 2003.

[14] E. Deelman and B. Szymanski. Dynamic load balancing in parallel discrete event simulation for spatially explicit problems. *SIGSIM Simul. Dig.*, 28(1), 1998.

[15] R. Dobrescu, S. Taralunga, and S. Mocanu. Web traffic simulation with scale-free network models. In *AIC'07: Proc. of the 7th Conference on 7th WSEAS International Conference on Applied Informatics and Communications*. WSEAS, 2007.

[16] A. Fabbri and A. Boukerche. Partitioning parallel simulation of wireless networks. In *Proc. of Winter Simulation Conference*. IEEE Press, 2000.

[17] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the Internet topology. *SIGCOMM*, Aug-Sept. 1999.

[18] R. Fujimoto. *Parallel and Distributed Simulation Systems. Wiley & Sons*, 2000.

[19] G. D'Angelo and M. Bracuto. Distributed simulation of large scale and detailed models. *To appear, International Journal of Simulation and Process Modelling (IJSPM)*, 2009.

[20] B. Gan, Y. Low, S. Jain, S. Turner, W. Cai, W. Hsu, and S. Huang. Load balancing for conservative simulation on shared memory multiprocessor systems. In *PADS '00: Proc. of the fourteenth workshop on Parallel and distributed simulation*. IEEE, 2000.

[21] B. Garbinato, D. Rochat, and M. Tomassini. Impact of scale-free topologies on gossiping in ad hoc networks. In *NCA*. IEEE Computer Society, 2007.

[22] Hyper-Threading Technology. http://www.intel.com/technology/platform-technology/hyper-threading/index.htm, 2009.

[23] IEEE 1516 Standard, Modeling and Simulation (M&S) High Level Architecture (HLA), 2000.

[24] R. J. Short and L. Kleinrock. Mobile wireless network system simulation. *Wireless Networks*, 1(4), 1995.

[25] D. Jefferson. Virtual time. *ACM Transactions Program. Lang. Syst.*, 7(3), 1985.

[26] M. Jelasity, A. Montresor, and O. Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM Transaction Computer Systems*, 23(3), 2005.

[27] H. Jeong, S. Mason, A.-L. Barabási, and Z. Oltvai. Lethality and centrality in protein networks. *Nature*, 411, 2001.

[28] R. K. Kincaid and N. M. Alexandrov. Scale-free networks: A discrete event simulation approach. In *International Conference on Computational Science (1)*, 2005.

[29] B. Kumova. Dynamically adaptive partition-based data distribution management. In *PADS '05: Proc. of the 19th Workshop on Principles of Advanced and Distributed Simulation*. IEEE, 2005.

[30] F. Liljeros, C. Edling, L. Amaral, H. Stanley, and Y. Aberg. The web of human sexual contacts. *Nature*, 411, 2001.

[31] B. Logan and G. Theodoropoulos. The distributed simulation of multi-agent systems. In *Proc. of the IEEE*, 2001.

[32] J. Misra. Distributed discrete event simulation. *ACM Computing Surveys*, 18(1), 1986.

[33] M. E. J. Newman. The structure and function of complex networks. *SIAM Review*, 45, 2003.

[34] P. Peschlow, T. Honecker, and P. Martini. A flexible dynamic partitioning algorithm for optimistic distributed simulation. In *PADS '07: Proc. of the 21st International Workshop on Principles of Advanced and Distributed Simulation*. IEEE, 2007.

[35] D. J. Price. Networks of scientific papers. *Science*, 149(3683), July 1965.

[36] G. Riley and M. Ammar. Simulating large networks: How big is big enough? In *Proc. of First International Conference on Grand Challenges for Modeling and Simulation*, Jan 2002.

[37] R.L. Bagrodia and R. Meyer. PARSEC: A parallel simulation environment for complex system. *IEEE Computer*, 31(10), 1998.

[38] M. Shanaker, R. Padman, and W. Kelton. Efficient distributed simulation through dynamic load balancing. *IIE Transactions*, 33(3), 2001.

[39] T. Som and R. Sargent. Model structure and load balancing in optimistic parallel discrete event simulation. In *PADS '00: Proc. of the fourteenth workshop on Parallel and distributed simulation*. IEEE, 2000.

[40] B. K. Szymanski, A. Saifee, A. Sastry, Y. Liu, and K. Madnani. Genesis: a system for large-scale parallel network simulation. In *PADS '02: Proc. of the sixteenth workshop on Parallel and distributed simulation*. IEEE, 2002.

[41] S. Verma and W. T. Ooi. Controlling gossip protocol infection pattern using adaptive fanout. In *ICDCS '05: Proc. of the 25th IEEE International Conference on Distributed Computing Systems*. IEEE, 2005.

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5672
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5672