SimGrid: a Generic Framework for Large-Scale Distributed Experiments

Henri Casanova
ICS Department,
University of Hawai'i at Manoa
Honolulu, USA

Arnaud Legrand
CNRS, MESCAL project
Laboratoire LIG
Grenoble, France

Martin Quinson
Nancy University,
LORIA
Nancy, France

Abstract

Distributed computing is a very broad and active research area comprising fields such as cluster computing, computational grids, desktop grids and peer-to-peer (P2P) systems. Unfortunately, it is often impossible to obtain theoretical or analytical results to compare the performance of algorithms targeting such systems. One possibility is to conduct large numbers of back-to-back experiments on real platforms. While this is possible on tightly-coupled platforms, it is infeasible on modern distributed platforms as experiments are labor-intensive and results typically not reproducible. Consequently, one must resort to simulations, which enable reproducible results and also make it possible to explore wide ranges of platform and application scenarios.

In this paper we describe the SimGrid *framework, a simulation-based framework for evaluating cluster, grid and P2P algorithms and heuristics. This paper focuses on* SimGrid *v3, which greatly improves on previous versions thanks to a novel and validated modular simulation engine that achieves higher simulation speed without hindering simulation accuracy. Also, two new user interfaces were added to broaden the targeted research community. After surveying existing tools and methodologies we describe the key features and benefits of* SimGrid.

1 INTRODUCTION

Distributed computing has become mainstream and arises today in a wide variety of domains for a wide range of applications. These include parallel scientific simulations running on clusters, large-scale "grid" applications running on platforms aggregating high-end resources across institutions, scientific applications running on "desktop grids" that harness the idle cycles of individually owned computers, multi-tier business applications based on Web services, and peer-to-peer applications for content dissemination and sharing. The number of such applications in production is rapidly growing. Alongside these development activities, distributed computing is an extremely active and broad research area covering distributed algorithms and protocols, distributed resource and content management, and resource and application scheduling. Many challenges remain for improving distributed applications (better performance, better resilience to faults, better scalability, etc.) and new challenges constantly arise due to the joint evolution of technology and of applications.

A key issue in distributed computing is to scientifically assess the quality of several solutions with respect to a particular metric (task throughput achieved by a scheduling heuristic, probability of availability of a service, response time of lookup queries, etc). In some (rare) cases this assessment can be addressed solely via theoretical analysis. Unfortunately, in most cases assessment can at best be obtained for very stringent and ultimately unrealistic assumptions regarding the underlying platform and/or the application. Therefore, most research results in these areas are obtained via empirical evaluation through experiments.

An obvious approach for obtaining valid experimental results is to conduct experiments on production platforms, or at least on large testbeds. Unfortunately, this often proves infeasible. Real-world platforms may not be available for the purpose of experiments, so as not to disrupt production usage. Results are often non-reproducible due to resource dynamics (e.g., time-varying and non-deterministic network usage, unpredictable host failures). Even if a stable platform is available, experiments can only be conducted for the platform configuration at hand, which may not be sufficient to gain all necessary insight in the relative effectiveness of various distributed system or algorithm designs. Finally, experiments on real-world platforms may be prohibitively time consuming especially if large numbers of experiments are needed to explore many scenarios with reasonable statistical significance. Given all these difficulties, while researchers always strive to obtain some experimental results in real-world systems, the majority of published results are obtained in simulation.

Simulation has been used extensively in several areas of computer science for decades, e.g., for microprocessor design and network protocol design. In these areas, several widely used and acknowledged simulation frameworks are available. By comparison, the use of sound simulation frameworks for distributed applications on distributed computing platforms is not as developed, certainly without any standard simulation tool (although network simulation is a key component of distributed application simulation). In this paper we describe a comprehensive simulation framework, SimGrid, for the simulation of distributed applications on distributed platforms. Our goal is to describe the salient capabilities of SimGrid and explain how they allow users to perform simulations for a wide range of applications and platforms. Some of the key features of SimGrid are:

- A scalable and extensible simulation engine that implements several validated simulation models, and that makes it possible to simulate arbitrary network topologies, dynamic compute and network resource availabilities, as well as resource failures;
- High-level user interfaces for distributed computing *researchers* to quickly prototype simulations either in C or in Java;
- APIs for distributed computing *developers* to develop distributed applications that can seamlessly run in "simulation mode" or in "real-world mode."

This paper is organized as follows. In Section 2 we discuss related work. Section 3 provides an overview of SimGrid, while Section 3.1 describes the base simulation engine. Section 3.2 describes the abstractions and APIs provided by SimGrid for quick simulation prototyping. Section 3.3 describes two components that make it possible to develop applications that can run seamlessly in simulation on in the real world. Section 4 provides some insights on quality of SimGrid-based simulations, in term of speed, scalability and accuracy. Finally, Section 5 concludes with perspectives on future work.

2 Background and Related Work

2.1 Methodologies

In this section, we review some of the numerous existing tools for conducting large-scale distributed computing experiments (a summary is given in Table 1). These tools can be sorted in four main categories: experimental platforms, emulators, network simulators and application simulators. More specifically, they can be classified depending on the following criteria:

- *Resource models:* Platforms contain resources of different types. For simulation purposes one typically considers *CPU* and *network* resources, and more rarely *disk* resources. For each resource type a tools may ignore it or implement a model for the performance delivered by the resources to the simulated application(s). Proposed models can be sorted in four categories (even if the boundaries are somewhat blurred): (i) very abstract *mathematical simulation* that are based solely on equations; (ii) *discrete-event simulation* (d.e.s.) that models the system as a set of dependent actions and events; (iii) *virtualization* that consists in intercepting and mediating resource usage on-the-fly; and (iv) *real execution* on real-world platforms without modifications.
- *Application model and requirements:* Most tools require the simulated algorithms to be written using a specific API in a specific language. Some emulation solutions even require full-fledge implementations.
- *Controls of experimental settings:* To conduct reproducible experiments, it is mandatory to control and configure the experimental settings, although some tools fail to provide such control.
- *Scalability:* The scale of the target platform varies depending on the research community. For example, P2P researchers study millions of nodes while cluster computing researchers study hundreds to thousands of nodes. Table 1 reports the maximum number of nodes that each tool can run, according to results in the literature or our own experience. This is only an upper limit, and other parameters (e.g., the execution speed) would be needed for a comprehensive comparison (which is beyond the scope of this paper).

2.2 Experimental Platforms

The main advantage of experimental platforms such as **Grid'5000** [4] or **PlanetLab** [8] is that results obtained on these systems are inherently *believable* since unbiased by simulation models. However, the scalability offered is limited since the Grid'5000 is for example constituted of "only" 1,500 nodes (4,000 cores) nationwide across France while PlanetLab comprises about 850 nodes world-wide. Moreover, PlanetLab does not allow to control the experimental settings, making it impossible to run back-to-back experiments under identical conditions.

2.3 Emulators

Emulation solutions let unmodified applications run in a specific environment where relevant system calls are intercepted and mediated. For example, **ModelNet** [20] models the network very accurately by injecting the intercepted traffic into a specific LAN and slowing down packet movements to mimic a target emulated platform precisely. Accuracy is at the price of very specific and constraining requirements on the LAN in terms of hardware and deployed OS. Finally, this tool only models the network, ignoring the CPU and disk resources, and does not scale easily beyond hundreds of nodes. **MicroGrid** [21] provides a lighter and more generic emulation solution. As with ModelNet, some system calls are intercepted and the application is slowed down to mimic execution on a target emulated platform. Differences with ModelNet are that computation times are taken into account and that communication times are computed via a network simulator similar to SSFNet [9].

2.4 Network Packet-Level Simulators

Over the years, the networking community has developed several network simulators. The most famous one is certainly **ns-2** [1], which implements a very large collection of protocols and queuing models. **SSFNet** [9] is another such simulator, providing a Java API for discrete-event simulation based on the SSF standard. **GTNetS** [18] benefits from a layered design (i.e., close to real stacks) and enjoys an impressive scalability up to 177,000 nodes.

The main problem with these simulators for our purpose is that they were designed by and for the networking community. Consequently, they track the movement of each and every packet in data flows. This makes them less adapted to application simulations where effects (i.e., communication times) are more important than causes (e.g, packets movements). The inordinate level of accuracy of fine grain d.e.s. may hinder performance (c.f. §4). Another important limitation of course is that these tools cannot simulate other resources than network resources.

2.5 Application Simulators

As a result of the difficulties and limitations of emulators and packet-level simulators, the distributed computing community has developed several application simulators in recent years. In fact, [15] reports that out of

	CPU	Disk	Network	Application	Requirement	Settings	Scale
Grid'5000 [4]	direct	direct	direct	direct	access	fixed	< 5000
PlanetLab [8]	virtualize	virtualize	virtualize	virtualize	none	uncontrolled	< 850
ModelNet [20]	-	-	emulation	emulation	lot material	controlled	100 nodes/real host
MicroGrid [21]	emulation	-	fine d.e.s.	emulation	none	controlled	few 100
ns2 [1]	-	-	fine d.e.s.	coarse d.e.s.	C++ and Tcl	controlled	<1,000 [18]
SSFNet [9]	-	-	fine d.e.s.	coarse d.e.s.	Java	controlled	<100,000 [18]
GTNetS [18]	-	-	fine d.e.s.	coarse d.e.s.	C++	controlled	<177,000 [18]
ChicSim [17]	coarse d.e.s.	-	fine d.e.s.	coarse d.e.s.	C	controlled	thousands
OptorSim [2]	coarse d.e.s.	amount	math/d.e.s.	coarse d.e.s.	Java	controlled	few 100
GridSim [5]	coarse d.e.s.	fine d.e.s.	fine d.e.s.	coarse d.e.s.	Java	controlled	few 100
PlanetSim [11]	-	-	constant time	coarse d.e.s.	Java	controlled	100,000 [15]
PeerSim [12]	-	-	-	state machine	Java	controlled	1,000,000 [15]
SimGrid	coarse d.e.s.	-	math/d.e.s.	d.e.s./emul	C or Java	controlled	few 10 000

Table 1: Comparison of existing tools for experimental large-scale distributed computing research.

141 papers about P2P and based on simulation, 30% use a custom simulation tool while half of them do not even report which simulation tool was used! Most of these tools only intend to be used by their own developers, and even when they are released to the public they target a very specific community. **ChicSim** [17] and **OptorSim** [2] are specifically designed to study data replication on grids. **GridSim** [5] was initially intended for grid economy, even if it became used in other areas of grid computing. **PlanetSim** [11] and **PeerSim** [12] are for the simulation of P2P applications. Likewise, the **SimGrid** project started as a community-specific project but since then its scope has been generalized, as seen in §3.

The above differences in goals and scope induce large differences in design. P2P tools such as PlanetSim and PeerSim aim at extreme scalability. They thus do not model CPU and disk since the classical performance metric in the P2P domain is message count. PlanetSim assumes a constant communication duration for each node pair while PeerSim completely ignore communication cost: time is merely discretized into steps at which each node can perform actions (send or receive messages).

Grid tools (ChicSim, OptorSim, GridSim and SimGrid) seek a compromise between execution speed and simulation accuracy because the main performance metric in the grid domain is application makespan. The CPU models are macroscopic: tasks costs are measured in MFlops while computer power is measured in MFlop/s. Only two tools model the disk: OptorSim does not take access time into account, but only available space; GridSim provides a fine grain model with latency, seek time and max transfer rate. For the network, ChicSim and GridSim mimic the flow fragmentation into packets that happens in real network, but they do not take TCP flow management mechanisms into account. This approach (called *wormhole*) makes them potentially as slow as packet-level simulators, but not quite as accurate. OptorSim and SimGrid rely on analytical models of TCP where flows are represented as flows in pipes [14]. This enables much faster simulations while allowing a reasonable level of accuracy (c.f. §4). Unfortunately, the bandwidth sharing algorithm used by OptorSim is clearly flawed when the platform is not homogeneous: the bandwidth share that each flow receives on a congested network link depends only on the number of flows using this link. This does not take into account the fact that some of these flows may be limited by other links in their path, preventing them to use the whole share on the aforementioned congested link. In such case, the unused link share is wasted while in the real-world it would be split between other (not otherwise limited) flows. This blatant shortcoming is even documented in the README file of the OptorSim distribution, but not fixed in the last release (v2.1).

This abundance of tools is problematic. First, it makes it difficult to compare the results in scientific papers, which is clearly a vexing problem for any scientific discipline. Then, even when simulation tools are publicly released, very few of them are sufficiently documented and maintained over time. Moreover, it usually proves difficult to use these tools outside of the specific domain for which they were designed. Finally, as exemplified by the unfortunate OptorSim bandwidth sharing algorithm, the simulators must be carefully validated.

3 The SimGrid Framework

The SimGrid project was initiated in 1999 to allow the study of scheduling algorithms for heterogeneous platforms. SimGrid v1 [6] made it easy to prototype scheduling heuristics and to test them on a variety of applications (expressed as task graphs) and platforms.

In 2003, SimGrid v2 [7] extended the capabilities of its predecessor in two major ways. First, the realism of the a simulation engine was improved by transitioning from a wormhole model to an analytical one. Second, an API was added to study non-centralized scheduling and other kind of concurrent sequential processes (CSPs).

This paper focuses on SimGrid v3, whose v3.0 version was released in 2005. v3.3, which is expected in 2008 and already available from the public SVN repository, adds many new features with respect to the previous versions. The simulation engine was completely rewritten, leading to better modularity, speed, and scalability. It now supports dynamic resource availabilities and failures. Two user interfaces were added to allow the use of the software in new contexts. The current SimGrid software stack with its relevant components is depicted in Figure 1.

SimGrid offers four user interfaces (detailed in §3.2 and §3.3): **SimDag** is the descendant of SimGrid v1 and is designed for the investigation of scheduling heuristics for applications as task graphs. **MSG** is the interface introduced in SimGrid v2 to study CSPs. **GRAS** allows to use SimGrid as a development lab for real distributed applications. **SMPI** enables the direct simulation of MPI

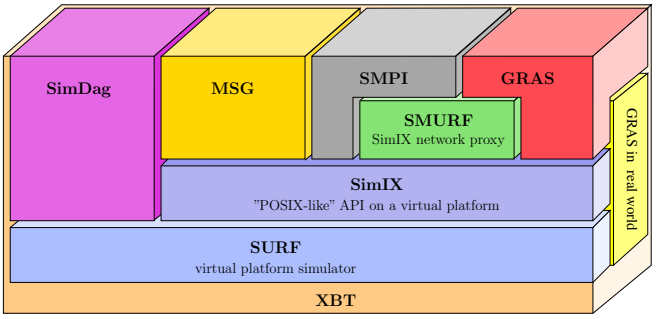

Figure 1: SimGrid components overview.

applications.

XBT is a "toolbox" module used throughout the software, which is written in ANSI C for performance. It implements classical data containers, logging and exception mechanisms, and support for configuration and portability. **SURF** is the code-name of the simulation engine, described in §3.1. **SimIX** is an internal module that provides a POSIX-like API on top of SURF, thus easing the development of simulation APIs that implement the abstraction of multiple concurrent processes. For instance, it would allow the development of openMP- or BSP-like user interfaces. The purpose of the **SMURF** module (at an early stage of development at the time this article is being written) is to allow the distribution of simulated processes over a cluster, harnessing the memory of several computers. This would allow to improve the scalability of SimGrid, which is currently limited by memory (c.f. §4.2).

3.1 Simulation Kernel and Models

SURF is the SimGrid simulation kernel. It was designed with two main goals in mind. First, it must be highly modular to allow the implementation (and comparison) of several resource models. In addition, as it constitutes the basis of the whole SimGrid framework, SURF must be carefully optimized so as not to hinder simulation speed.

From a conceptual standpoint, a running simulation consists of a set of resources that deliver some capability and of an application that wishes for these resources to accomplish a set of actions (network links must convey bits, CPUs must perform computation, etc.). The main goal of the user interfaces such as SimDag or MSG is to provide a convenient way for users to express these actions. The main goal of the SURF component is to compute the completion times of these actions.

To achieve the above, SURF "asks" the model governing each resource when an ongoing action will be completed by the resource and computes the minimum of all completion dates. (Note that an action may use more than one resource.) SURF advances the simulated time to this date (updating the action and resource states) and then informs the user interface of the terminated action(s). User (simulation) code thus gets a chance to execute. The initiation of a new action (communication or computation) blocks the user code again and launches SURF into another such simulation cycle.

This design separates resource modeling from the main simulation loop. Version v3.3 of the software comes with five distinct resource models. The default one (*CM02*) implements a MaxMin sharing strategy [3] for both the network and the CPU. *Vegas* and *Reno* implement the corresponding TCP algorithms thanks to analytical models inspired from [14]. *PtaskL07* models the execution of parallel tasks. This modular design even enabled us to implement a pseudo-model in which the network is simulated thanks to the GTNetS packet-level simulator (see [10] for more details).

3.2 User Interfaces for Researchers

SimDag SimDag allows to prototype and simulate scheduling heuristics for applications structured as task graphs of (possibly parallel) tasks. With this API one can create tasks, add dependencies between tasks, retrieve information about the platform, schedule tasks for execution on particular resources, and compute the DAG execution time.

MSG This interface was added into SimGrid v2 to allow the study of CSP applications. While initially intended for studying scheduling algorithms, it proved perfectly usable in other contexts, such as desktop grids [13] and in time became the most widely used SimGrid API. For this reason, this interface is frozen: existing API functions will not changed (but new functions are added to fulfill new needs). This is to ensure that code written with MSG remains functional with subsequent releases of SimGrid. Version 3.3 introduces Java bindings to the MSG API (called **jMSG**), allowing user reluctant to program in C to still use SimGrid.

3.3 User Interfaces for Developers

GRAS (Grid Reality And Simulation) This API, added in v3.0, allows the development of distributed applications (e.g., a resource information service) within the simulator, but that are then seamlessly deployable on real-world platforms without code modification. To enable this, the same API is implemented twice (once for the simulated world, once for the real world), and user code simply has to be linked against the appropriate library. This is depicted on the right face of the software layer block in Figure 1. GRAS is described in more details in [16].

SMPI This module is currently under finalization for a release in version 3.3. It will allow the simulation of unmodified MPI application [19] by intercepting MPI primitives in a manner similar to the MicroGrid approach [21].

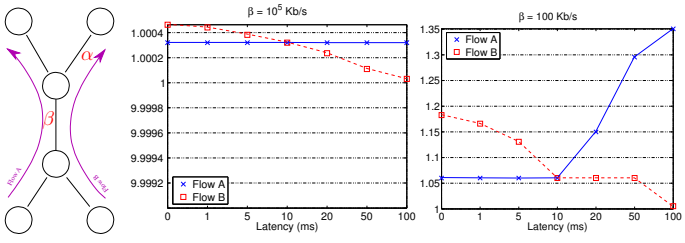

Figure 2: Relative error of SimGrid over GTNetS on a dogbone topology.

4 Experimental Evaluation

4.1 Simulation Accuracy

While it is far beyond the scope of this article to present comprehensive simulation validation results, we provide a few highlights and point to other published material. The work in [10] compares the MaxMin network model of SimGrid to that of the GTNetS packet-level

simulator in several scenarios. For the sake of discussion it is assumed that GTNetS results are 100% correct and any discrepancy between the SimGrid results and the GTNetS results is considered SimGrid "error." We leave comparison of both GTNetS's network model and of SimGrid's MaxMin network model to real networks, as well as the evaluation of other network models implemented in SimGrid, for future work.

Figure 2 presents the relative error of SimGrid when compared to GTNetS for a classical dogbone topology. The latency of one of the "leg" links is varied, thus changing the RTT of one of the flows. The left graph shows results for a non-limiting shared link (in which case flow bandwidths are limited by flow RTTs because of TCP congestion control mechanisms). In this case, the SimGrid error is very low (below 0.5%). The right graph is for the case when the limiting resource is the bandwidth of the shared link. In this case the error is higher (up to 35%).

Overall, experiments presented in [10] show that:

- Since SimGrid does not account for TCP's slow-start mechanism, its accuracy is poor for short messages;
- When a flow's bandwidth is limited by the flow's RTT, the SimGrid error is very low (below 1%);
- When a flow's bandwidth is limited by the actual link bandwidth, there seems to be a fixed error of roughly 5%. We are currently calibrating the simulator to address this problem;
- When network congestion rises, SimGrid leads to optimistic results (i.e., flows receive more bandwidths than they do with GTNetS). Amongst other causes, this comes from the fact that packet dropping by overloaded routers is not modeled in SimGrid. The error seems to be heavy-tailed: even in an highly contented network comprising 50 nodes interconnected by links of about 50Mb/s exchanging 200 flows of 100Mb each, we observe that for 50% of the flows the error is below 10%, while for 90% of them, the error remains below 50%. Some outliers however exhibit error up to 100%.

4.2 Simulator Scalability and Speed

	GTNetS		SimGrid	
# of flows	Running time	slowdown	Running time	slowdown
10	0.661s	0.856	0.002s	0.002
100	7.649s	7.468	0.137s	0.140
200	15.705s	11.515	0.536s	0.396

(a) 1Mb flows

	GTNetS		SimGrid	
# of flows	Running time	slowdown	Running time	slowdown
10	65s	0.92	0.001s	0.00002
100	753s	8.08	0.138s	0.00142
200	1562s	12.59	0.538s	0.00402

(b) 100Mb flows

Table 2: Comparison of SimGrid and GTNetS execution times.

4.2.1 Network simulation performance

Table 2 compares the execution time of GTNetS and SimGrid when simulating various numbers of flows. The corresponding slowdown is also reported. For example, GTNetS needs 0.661 seconds to simulate 10 flows that transfer 1Mb while SimGrid needs 0.002 seconds to do the same. From these values, we conclude that the running time of GTNetS is linear in both the number of flows and their sizes while the running time of SimGrid is only linear in the number of flows.

4.2.2 Simulation scalability

The classical metric for a simulator's scalability is the number of simulated nodes/processes that are allowed to start. In our case, this is only a matter of available memory since nodes are not mapped into (p)threads by default, but into UNIX98 contexts. This removes any limitation on the amount of threads from libc or the kernel and has allowed us to run simulation with up to 2,000,000 simulated processors (without swapping) on a computer with 16Gb of memory.

We feel however that computation time is more representative of usability in practice. Table 3 shows how the simulator's running time increases when the number of simulated actions increases. The simulated application is a classical master/worker application in which the number of tasks dispatched from the master to the workers varies between 1,000 and one million (on a 2Ghz 32-bit Laptop with 1Gb of memory). The first case allows us to quantify the simulator startup and setup time. The simulation only comprise 100 nodes, but the relevant parameter here is the number of actions injected in the simulator. Dispatching 100,000 tasks over 100 or 1,000 nodes takes about the same time.

# tasks	Native version	Java version
1,000	0.7s	0.5s
10,000	1.7s	2.5s
100,000	9.6s	23s
1,000,000	96s	240s

Table 3: Simulation of the master/worker example.

The performance is linear in the number of tasks to dispatch. The performance difference between the native and Java versions can be explained by the fact that we have to use Java threads and synchronization mechanisms. This is because mixing threads and UNIX contexts leads to perilous technical difficulties since the JVM is itself multithreaded. The performance loss is thus a comparison of Java threads (likely implemented using native pthreads) and UNIX contexts.

5 Conclusion

In this paper, we have reviewed existing tools for experimental large-scale distributed computing research. The use of real-world platforms proves time- and labor-intensive while not allowing reproducibility. Emulators, although attractive, often prove too cumbersome and are rarely used by researchers in our area. Packet-level simulators from the networking community prove poorly adapted to our purpose. Consequently, an inordinate number of in-house, short-lived and highly specialized simulation tools have been developed to fulfill the need of the community. Expectedly, these tools are difficult to use by others in other contexts than the ones for which they were intended. Moreover, tool proliferation hinders

the comparison of (often unreproducible) published results.

We presented the SimGrid simulation framework whose goal is to provide a generic evaluation tool for large-scale distributed computing. We presented its main components: two APIs for *researchers* who study algorithm and need to prototype simulations quickly, and two for *developers* who can develop applications in the comfort of the simulated world before deploying them seamlessly in the real world. SimGrid employs a modular simulation kernel that supports the addition and use of new resource models without changes in the user code. We used this feature ourselves to implement several simulation models and even to integrate the GTNetS packet-level simulator with SimGrid.

SimGrid consists of 30,000 lines of GPL'ed code. It is freely available from its Web page[1] and comes with all relevant information as well as with several example programs and tutorials. SimGrid uses an extensive regression testing suite as well as set of automatic compilation daemons ensuring a reasonable level of software quality. It is ported to Linux, Windows, Mac OS X and AIX.

SimGrid is a very active project, both in terms of research and development, and we envision many directions for future work. We first plan to add a model for disk resources. We also plan to improve usability in the P2P domain by further extending scalability. A first solution of course would be to implement an efficient but simplistic network model that returns constant communication times. Another direction currently under investigation is the ability to dispatch simulated nodes over several physical machines to go beyond the memory limit of a single computer.

References

[1] The Network Simulator (ns2). http://nsnam.isi.edu/nsnam/.

[2] W. H. Bell, D. G. Cameron, L. Capozza, A. P. Millar, K. Stockinger, and F. Zini. OptorSim - A Grid Simulator for Studying Dynamic Data Replication Strategies. *International J. of High Performance Computing Applications*, 17(4), 2003.

[3] D. P. Bertsekas and R. G. Gallager. *Data Networks*. Prenctice Hall, 1987.

[4] R. Bolze, F. Cappello, E. Caron, M. Daydé, F. Desprez, E. Jeannot, Y. Jégou, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, P. Primet, B. Quetier, O. Richard, E.-G. Talbi, and I. Touche. Grid'5000: A Large Scale And Highly Reconfigurable Experimental Grid Testbed. *International Journal of High Performance Computing Applications*, 20(4):481–494, 2006.

[5] R. Buyya and M. Murshed. GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing. *J. of Concurrency and Computation: Practice and Experience (CCPE)*, 14(13-15), Decembre 2002.

[6] H. Casanova. Simgrid: A toolkit for the simulation of application scheduling. In *IEEE International Symposium on Cluster Computing and the Grid (CCGrid)*, page 430, Los Alamitos, CA, USA, 2001. IEEE Computer Society.

[7] H. Casanova, A. Legrand, and L. Marchal. Scheduling Distributed Applications: the SimGrid Simulation Framework. In *IEEE International Symposium on Cluster Computing and the Grid (CCGrid)*, may 2003.

[8] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. PlanetLab: an overlay testbed for broad-coverage services. *ACM SIGCOMM Comput. Commun. Rev.*, 33(3):3–12, 2003.

[9] J. H. Cowie, D. M. Nicol, and A. T. Ogielski. Modeling the global internet. *Computing in Science and Engineering*, 1(1):42–50, 1999.

[10] K. Fujiwara and H. Casanova. Speed and Accuracy of Network Simulation in the SimGrid Framework. In *Workshop on Network Simulation Tools (NSTools)*, 2007.

[11] P. García, C. Pairot, R. Mondéjar, J. Pujol, H. Tejedor, and R. Rallo. Planetsim: A new overlay network simulation framework. In *Software Engineering and Middleware, SEM 2004*, volume 3437 of *LNCS*, pages 123–137, Linz, Austria, March 2005.

[12] M. Jelasity, A. Montresor, G. P. Jesi, and S. Voulgaris. PeerSim. http://peersim.sourceforge.net/.

[13] D. Kondo. Simboinc. http://simboinc.gforge.inria.fr/.

[14] L. Massoulié and J. Roberts. Bandwidth Sharing: Objectives and Algorithms. In *INFOCOM*, volume 3, pages 1395–1403, 1999.

[15] S. Naicken, A. Basu, B. Livingston, and S. Rodhetbhai. Towards yet another peer-to-peer simulator. In *Proc. Fourth International Working Conference Performance Modelling and Evaluation of Heterogeneous Networks (HET-NETs '06)*, September 2006.

[16] M. Quinson. GRAS: a Research and Development Framework for Grid and P2P Infrastructures. In IASTED, editor, *International Conference on Parallel and Distributed Computing and Systems*, 2006.

[17] K. Ranganathan and I. Foster. Decoupling computation and data scheduling in distributed data-intensive applications. In *Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing*, page 352, Washington, DC, USA, 2002. IEEE Computer Society.

[18] G. F. Riley. The Georgia Tech Network Simulator. In *ACM SIGCOMM workshop on Models, Methods and Tools for Reproducible Network Research*, pages 5–12, 2003.

[19] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra. *MPI the complete reference*. The MIT Press, 1996.

[20] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostić, J. Chase, and D. Becker. Scalability and accuracy in a large-scale network emulator. In *OSDI '02: Proceedings of the 5th symposium on Operating systems design and implementation*, pages 271–284, New York, NY, USA, 2002. ACM.

[21] H. Xia, H. Dail, H. Casanova, and A. A. Chien. The MicroGrid: Using Online Simulation to Predict Application Performance in Diverse Grid Network Environments. In *Second International Workshop on Challenges of Large Applications in Distributed Environments (CLADE)*, Los Alamitos, CA, USA, 2004. IEEE Computer Society.

[1] http://simgrid.gforge.inria.fr/