Scheduling Distributed Applications: the SimGrid Simulation Framework

Arnaud Legrand Loris Marchal

Laboratoire de l'Informatique du Parallélisme
École Normale Supérieure de Lyon

Henri Casanova

Dept. of Computer Science and Engineering
San Diego Supercomputer Center
University of California at San Diego

Abstract— **Since the advent of distributed computer systems an active field of research has been the investigation of *scheduling* strategies for parallel applications. The common approach is to employ scheduling heuristics that approximate an optimal schedule. Unfortunately, it is often impossible to obtain analytical results to compare the efficacy of these heuristics. One possibility is to conducts 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 (i.e. *Grids*) as it is labor-intensive and does not enable repeatable results. The solution is to resort to *simulations*. Simulations not only enables repeatable results but also make it possible to explore wide ranges of platform and application scenarios.**

In this paper we present the SimGrid framework which enables the simulation of distributed applications in distributed computing environments for the specific purpose of developing and evaluating scheduling algorithms. This paper focuses on SimGrid v2, which greatly improves on the first version of the software with more realistic network models and topologies. SimGrid v2 also enables the simulation of distributed scheduling agents, which has become critical for current scheduling research in large-scale platforms. After describing and validating these features, we present a case study by which we demonstrate the usefulness of SimGrid for conducting scheduling research.

I. Introduction

Since the advent of distributed computer systems, the question of *scheduling* parallel applications has generate a very large body of work. The process of scheduling consists in assigning the tasks of an application to available resources, both in time and space, with as objective the minimization of a metric (e.g. execution time, throughput, resource utilization, monetary cost). Most scheduling problems are NP-hard. Even when an optimal solution to a scheduling problem can be found in polynomial time, it is often the case that small modifications of the underlying assumptions (e.g. addition of non-zero network latencies) render the problem NP-hard. As a result, a common approach is to use *heuristics* to approximate an optimal schedule. Heuristics have been developed by several researchers for different classes of scheduling problems While these heuristics have low complexity and can thus be used in practice. it is generally not possible to quantify their efficacy analytically. Consequently, scheduling heuristics must be compared by performing many experiments in many scenarios.

One approach is to perform experiments with real applications on real resources. However, modern computing platforms are increasingly distributed and often span multiple administrative domains [17]. Therefore, resource availability fluctuations makes it impossible to conduct repeatable experiments for relatively long running applications are problematic. Another problem is that the number of platform configurations that can be explored is limited. As a results of these difficulties with real experimentations, most researchers have resorted to discrete-event simulation (which we will refer to as "simulation").

Simulation has been used extensively as a way to evaluate and compare scheduling strategies as simulation experiments are configurable, repeatable, and generally fast. In spite of these advantages, we claim that there are two main limitations to the simulation methodology used for scheduling research. First, there is no simulation standard in the scheduling research community. Typically, researchers build "throw-away" simulators using a variety of programming languages and tools. This makes it very difficult for other researchers to reproduce results presented in papers, and therefore to compare results among papers. This prevents scientific advances and is in contrast to other research communities in which simulation standards have been adopted (e.g. networking, computer architecture). This lack of standard simulation procedure and software was somewhat justifiable when the simulation models in use were simplistic. However, traditional models and assumptions about computing plat-

forms are no longer valid for modern platforms, which leads us to the second limitation of simulations used in the scheduling literature. The simplistic network models used in most scheduling literature do not hold for modern computing platforms in which compute resources are connected over complex network topologies with complex link contention behaviors. The assumption that the behavior of the compute platform is perfectly predictable also needs to be revisited as modern platforms exhibit dynamic resource availabilities. Realistic simulation must make it possible to simulate such fluctuations as they cause uncertainty about the platform knowledge that a scheduler can obtain.

Consequently, there is a need for a simulation framework designed for conducting research on distributed application scheduling. To be useful this framework must meet the following objectives. It must provide *good usability* so that users can focus on scheduling research and not on simulation issues. It must make it possible to run *fast simulations* because enormous numbers of simulation experiments must be conducted to evaluate and compare scheduling algorithms. This framework must make it possible to build *configurable*, *tunable*, and *extensible* simulations. In particular, it should be possible to go from "traditional" simulation models to more realistic ones without having to rethink the implementation of the simulation. Finally, the simulation framework must be *scalable* and sustain simulations with tens of thousands of resources and application tasks. We claim that no available simulation framework currently matches these requirements.

To address these issues, we have designed the SimGrid simulation framework. This paper focuses on SimGrid v2, which greatly improves on the first version of the software [6]. Enhancements include improved and more realistic network models, means to import platform simulation models from platform monitoring tools, and improved APIs. Most importantly, a new software layer makes it easier to build complex simulation by providing higher-level abstractions. In particular, it provides capabilities for simulating distributed scheduling agents. This is becoming critical in scheduling research on large-scale platforms in which a centralized scheduler is a single point of failure and reduces scalability. SimGrid has already been used successfully for a number of research works [13, 35, 2, 3, 39, 34].

II. Background and Related Work

A large number of software tools have been developed for users to build and run simulations in many application domains. In this section we review those tools that could be employed to conduct research in the area of distributed application scheduling.

Several software libraries and environments provide ways to construct generic discrete-event simulations [31, 38, 28, 15, 18, 20]. Such generic-purpose tools could be used for simulating parallel applications on distributed platforms. An impediment is that the Application Programming Interfaces (APIs) provided by these tools are not tailored to the investigation of scheduling algorithms. The APIs are low-level, because generic, and a user who wishes to do research on scheduling algorithms is faced with the task of implementing many higher-level abstractions. Each researcher would probably build different such abstractions, thereby not solving the problem of simulation standardization identified in Section I. Besides, researchers in the area of scheduling seldom use these low-level simulation packages in current practice.

An area in which simulation is used extensively is networking research. In order to understand how large telecommunication networks scale and behave in various situations, several network simulators have been developed. Example of such simulators include NS [29, 1], DaSSF [23] or OMNeT++ [30]. These simulators focus on precise simulation of packets traveling on the network, rather than on the network behavior as it is experienced by an application. The main intent is to help identifying limitations of network protocols and help developing improvements. Although it is possible to perform application-level simulation with certain of these tools (e.g. with DaSSF [23] or OMNeT++ [30]), it is labor-intensive and a number of capabilities need to be implemented from scratch. Furthermore, due to their highly detailed simulation models, most network simulators induce long simulation times (e.g. they implement the TCP stack). This issue is exacerbated when one wishes to simulate "background" network traffic generated by other users/applications as one may then need to simulate thousands of connections at the packet-level. This is prohibitive for conducting the large numbers of simulation experiments mandated by scheduling research.

Another limitation of network simulators for our purpose is that they do not simulate the entire platform. Even though the network is a fundamental component of a distributed computing platform, other components are required for investigating scheduling algorithms: compute resources and application processes. A number of projects address the simulation of a complete distributed computing platforms for studying paral-

lel applications. We review here three representative projects: LAPSE, MicroGrid, and Albatross.

LAPSE (Large Application Parallel Simulation Environment) [10] is a tool designed to simulate parallel applications. The main goal is to make it possible to test the scalability of parallel application while running them only on a few processors. A focus of the project is precise network event simulation. The application code is instrumented so as to account for the number of instructions executed between two consecutive communication. LAPSE was originally targeted to the Intel Paragon architecture, but allows the simulation of other platforms. One important limitation is that the network model cannot capture contention and cannot easily model background network traffic.

Microgrid [33], a more recent project, targets the simulation of Grid environments. It follows somewhat the same philosophy as LAPSE in that it emulates large, complex, virtual platforms on small, physical platforms. The software virtualizes every resource of a Grid platform (memory, CPU, and network). Just as in LAPSE, this virtualization is achieved by trapping every relevant library call. In Microgrid's first version, the network was simulated through a modified version of the NS [1] simulator but the latest version uses DaSSF [23]. At the moment, Microgrid does not allow the simulation of variable, background network traffic.

The Albatross [21] project focuses on platforms that consist of multiple clusters or MPPs that are connected by wide-area networks (WANs). In the context of this project, work based on the Panda library [8] addresses the simulation of these platforms. Unlike LAPSE and Microgrid, the network is not simulated but emulated by slowing down LAN links to model WAN links. This makes it straightforward to dynamically modify network behavior during a simulation. A limitation is that heterogeneity is modeled only for WAN links.

In these three projects, portions of the application are effectively executed on an emulated architecture. The rationale is that emulation is realistic because it can capture more detailed and idiosyncratic behaviors. However, a consequence is that the ratio of simulated time to simulation time can be very large. This is prohibitive for comparing scheduling algorithms in tens of thousands of scenarios.

We conclude that no existing simulation framework satisfies the requirements that we have identified in Section I. In the rest of this paper we describe our work on SimGrid which addresses these requirements. The Bricks simulator [37] is related to SimGrid but targets the simulation of client-server applications and follows a different approach for implementing the simulation, which we discuss further in Section IV-C. Note that the GridSim [4] project is very related to our work in scope and intent. GridSim is is based on SimJava [20] and implements a number of abstractions to enable simulation of Grid environments. At the time this article is being written, we were unable to successfully install and test the GridSim software distribution, hence preventing us from comparing it with our work.

III. SimGrid v2.0

A. History

The first version of SimGrid was a discrete-event simulation toolkit. It provided a set of core abstractions and functionalities that can be used to easily build simulators for specific application domains and/or computing environment topologies. This allows the simulation of arbitrary performance fluctuations such as the ones observable for real resources due to background load. However, this first version lacked a number of abstractions (e.g. routing, scheduling agents). With SimGrid v2 we have added a new software layer to provide high-level abstractions and the software thus provides two interfaces:

SG : The original low-level toolkit presented in [6], by which the simulation is done in terms of explicitly scheduling tasks on some resources;

MSG : A simulator built using SG. This layer implements realistic simulations based on the foundational SG and is more application-oriented. Simulations are built in terms of communicating agents.

In all that follows, we focus on the MSG interface as we anticipate it will become the standard interface for the vast majority of SimGrid users. Also, "SimGrid" denotes version 2 of the software throughout.

B. Fundamental Concepts

SimGrid implements the following core abstractions:

Agent – An agent is an entity which makes scheduling decisions. An agent is defined by a code, private data, and the location at which it executes.

Location – A location (or host) is the place in the simulated topology at which an agent runs. Thus it is defined by a computational resource, a number of mailboxes that enable communication with other agents, and private data that can be only accessed by agents at the same location.

Task – A task is an activity of the simulated application and for now can be a computation and/or a data

transfer. A task is defined by an amount of computing, a data size, and private data.
Path – The low-level, original SG layer in the software did not provide any abstraction for message routing among locations. This made the task of the user arduous when simulating complex platforms. SimGrid now provides a routing abstraction so that the user (and the scheduler) can rely of a logical topology of the platform rather than targeting the physical topology directly. A path is an agglomeration of communication resources representing a set of physical network links. Locations are then interconnected through paths. The simulated application cannot access links directly (in the same way as a real application does not choose which routers its packets go through).
Channel – Communication between agents is embedded in the channel abstraction. A channel embodies the concept of communication ports opened by agents at locations.

With these abstractions, scheduling algorithms with SimGrid should always be described in terms of agents that run at locations and interact by sending, receiving, and processing simulated application tasks. Agent does not have direct access to paths but can send a task to another location using a channel. In fact, a location may have many mailboxes and a channel is then simply a mailbox number. So sending a task to a location using a channel amounts to transferring the task on a particular path, depending on the emitter location and on the destination, and to put it in a particular mailbox. We claim that these abstractions are sufficient to conduct scheduling research while isolating the user from details of the simulation that are not directly relevant to scheduling algorithms.

C. Building a Simulation with SimGrid

A SimGrid program always follows the following steps:
1. Definition of the code of each agent (i.e. modeling the application): the modeling of the agents is done with functions such as `MSG_task_get`, `MSG_task_put` or `MSG_task_execute` to handle tasks. Even if these functions are blocking, they are sufficient to encode any programming style (synchronous or asynchronous message passing, remote procedure call or process migration) since it is possible to create dynamically some new agents. Simple examples of resource selection and monitoring are provided but it is entirely free and has to be implemented by the user;
2. Creation of resources (i.e. modeling of the physical platform). It consist in defining hosts, links, and a routing table that specifies paths. It can be done either by hand using functions like `MSG_host_create`, `MSG_link_create`, `MSG_routing_table_set`,... or automatically by reading a platform description file (see section IV-B);
3. Creation and allocation of agents to locations (i.e. the deployment of the application). It is done with the function `MSG_process_create`.
4. Simulation can then be started with the function `MSG_main`. Different levels of traces can be enabled with the `MSG_set_verbosity` function.
The SimGrid software distribution provides documentation as well as a number of tutorial examples that illustrate these four steps.

IV. Platform Modeling with SimGrid

SimGrid provides several mechanisms for constructing simulated computing platforms. In this section we review the basic resource models, describe how realistic platform topologies can be generated, and describe how SimGrid simulates bandwidth-sharing.

A. Basic Resource Models

Low-level resource objects in SimGrid are *hosts* and *links*. A host is described by its computational speed relative to that of a reference host, and by its CPU availability (a value between 0 and 100%). A link is described by a latency and a bandwidth. By default, SimGrid uses the traditional models of task execution time equal to the computational cost divided by the computational speed, and of data transfer time equal to the latency plus the data size divided by the bandwidth. Note that the units of these values are not specified. Instead the user must ensure coherence between units of data size and bandwidth, and of task computational cost and computational speed. All values characterizing SimGrid resources can be either fixed, of varying according to a trace (a series of time-stamped values).

An important question is that of resource sharing among tasks of the simulation (including computational tasks and data transfer tasks). SimGrid implements three different sharing modes: (i) First In First Out (FIFO); (ii) First Ready First Out (FRFO); (iii) Shared. In the FIFO mode tasks on a resource execute in the order they were scheduled on that resource. In the FRFO mode, tasks that are ready for execution execute first. If there is a tie, then the task that was scheduled first executes first. In the Shared mode all ready tasks execute concurrently on a resource and SimGrid allows the user to implement fair sharing or

proportional sharing with task priorities. These different modes provide some flexibility and make it possible to implement simulations with traditional assumptions from the scheduling literature.

The strength of SimGrid is that the user can easily use the API to tune the behavior of the simulated environment. For instance, moving from resources with constant performance characteristics to resources with characteristics that vary according to traces can be achieve by just modifying a few parameters of the resource creation API functions.

The current trend in high-performance computing is to move to Grid platforms in which resources are interconnected over complex wide-area networks. For applications to benefit from such platforms it is necessary that novel scheduling approaches be designed that can accommodate complex networks. As a result, it is necessary to provide simulation tools that can capture the behavior of these networks and which can in turn enable new scheduling research. The default resource models in SimGrid described above do not fully enable the transition from traditional parallel computing to Grid computing scheduling research.

We identify two challenges that must be addressed to enable this transition. First, it must be possible for users to construct large simulated platforms that are representative of existing platforms. Second, it must be possible to simulate the complex network contention behaviors of applications executing on these platforms. In the next two section we describe how SimGrid addresses these two challenges and makes it easy for users to build realistic simulation models for studying scheduling on Grid platforms.

B. Modeling Grid Topologies

Most simulations reported in the scheduling literature have used either simple topology models (e.g. a ring or a star), representations of a real topology (e.g. the Arpanet), or randomly-generated flat topologies using a variation of Waxman's edge-probability function. Recently, complex randomly-generated hierarchical models have been used to better approximate the Internet's hierarchical structure. Many studies on Internet topologies are available [11, 5, 14, 27]. Using snapshots of the Internet, simple power-laws of the Internet topology have been derived, leading to several public-domain generators of "representative" Internet topologies [19, 26, 36].

While these generators constitutes an excellent starting point for building a realistic simulation model, they lack several critical elements for our purpose. Typically, there is no information concerning the traffic (e.g. the available bandwidth throughout time) and no information on the characteristics of network links. Common techniques involve the use of simulated sources of traffic using random laws which are seldom validated against real traffic. Finally, several parameters have to be hand-tuned to obtain realistic platforms. In summary, the "art" of building a realistic platform model requires both significant experience and good intuition.

The alternative is, rather than simulating traffic sources, to collect traffic traces from a real environment. These traces can then be replayed for conducting repeatable simulation experiments, which is needed for the investigation of scheduling algorithms. To enable this, SimGrid makes it possible for users to automatically *import* platform descriptions obtained with Effective Network View (ENV) [32]. ENV is a tool that discovers a variety of characteristics of a Grid testbed including the network and its effective topology. The obtained topology can be used to deploy NWS sensors and record real-time traces which are directly usable in SimGrid. The combination of ENV and NWS makes it possible to instantiate platform models which represent realistic platforms both in terms of topology and in terms of traffic, as demonstrated in [22]. These models can then be used as is, or as basis for generating realistic randomized collections of models for conducting large numbers of simulations.

C. Bandwidth Sharing Models in SimGrid

Modeling realistic topologies is a critical step forward to enable scheduling research for emerging computing platforms such as the Grid. An important question is then that of simulating contention among data transfers on these topologies. The default resource models in SimGrid as described in Section IV-A do not enable realistic simulation of network traffic on topologies in which paths between hosts use multiple network links. Indeed, the basic models implement a simple store-and-forward scheme, without any notion of packets or network pipelining. Furthermore, if the Shared resource sharing mode is used for network links, it is not clear how to assign priorities among competing transfers (it is well-known that bandwidth-sharing among TCP flows is not fair [9, 16, 25]).

A possibility is to simulate the network at the packet-level. While this is fully implemented in simulators like NS [29], it leads to prohibitive simulation times for our purpose. A simplified version of packet-level network simulation is is available in SimGrid as well as in the Bricks simulator [37]. The user can trade-off speed ver-

sus accuracy by tuning the packet size (large packet sizes are cheaper to simulate). This approach has two limitations. First, the actual network protocols are not modeled. Second, the trade-off between accuracy and simulation speed is difficult to quantify.

Another approach is to use macroscopic models of bandwidth-sharing and to implement these models directly in the simulation, thereby drastically shortening simulation times. Such models are typically derived via an analogy between network connections, or *flows*, and fluids in pipes while ignoring the packet granularity [24]. Several authors have proposed theoretical models for TCP traffic [9, 16, 25]. Building on these results, SimGrid implements an efficient algorithm for simulating TCP flows competing over multi-path routes. Due to lack of space we only give here the main ideas (see [7] for details). The algorithm first considers all links and determines bottleneck links for some flows. These flows are assigned bandwidth on these links inversely proportionally to their round-trip times. These flows consume this bandwidth end-to-end, and thus our algorithm reduces the bandwidth capacity of links traversed by all these flows in the network accordingly. This process is repeated until bandwidth has been allocated to all flows. We have proved the correctness of our algorithm and validated our model with the Network Simulator (NS) [29]. Bandwidth-sharing is completely transparent to the SimGrid user.

SimGrid makes it possible to define two types of links for the purpose of bandwidth-sharing: links on which the bandwidth is shared among flows of the simulated applications, and links on which it is not. On these latter links it is assumed that all competing flows of the simulation application achieve the same, constant data-transfer rate. The rationale is that, over a congested backbone, the contention due to interaction between flows of the simulated application is negligible compared to that of the many (thousands) flows going through the backbone. SimGrid computes bandwidth-sharing for topologies that contain both types of links. This makes it possible to build Grid models as sites interconnected by backbone links while each site has its own internal topology (see [7] for a discussion). We conjecture that such a model is simple enough to be instantiated easily while it capture the relevant network behaviors necessary for conducting meaningful Grid computing scheduling research. We are validating this conjecture in current work and will report on results in upcoming papers.

A key strength of SimGrid is that its simple API makes it possible to use a wide variety of platform models interchangeably without impacting the implementation of the scheduling algorithms. It is possible for every user to pick and choose which model is appropriate for his/her research and to increase the model complexity when warranted. The conjectural Grid model that we briefly described above is just one of the many possible models that can be implemented with SimGrid. The question of which Grid model is "right" is not answered here, but SimGrid provides framework for making progress towards an answer.

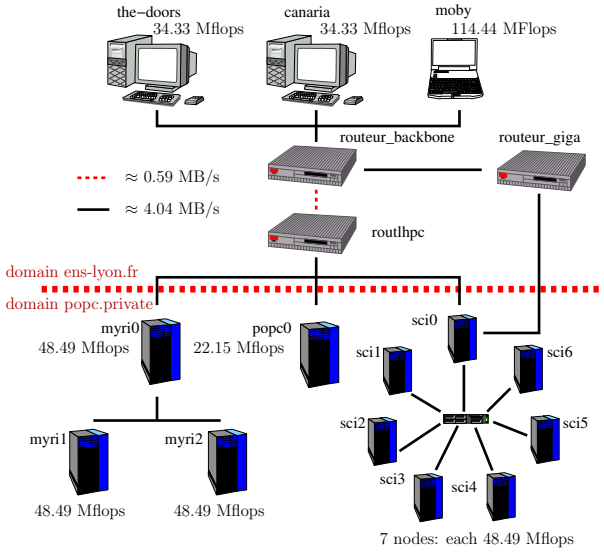

Fig. 1. Realistic platform imported from ENV.

V. Case Study

To better illustrate the capabilities and ease-of-use of SimGrid we present a small case study. The target application is a high-throughput master-worker application such as the ones being deployed in many global computing systems [12]. A *master* dispatches input data for computational tasks to p *workers* one at a time. The question we wish to answer is: "Given n independent, identical tasks that must be distributed to workers, how should the master fulfill requests for work from the workers?"

This is a classical scheduling problem and a straightforward strategy is to employ a greedy algorithm by which the master allocates tasks on a first-come first serve basis. We denote this strategy as *Work Queue*. Another approach is to maintain a "pool" of pending requests, $p/2$ such requests in our example, and to dispatch work according to priorities. Several criteria for prioritization are possible and in this case study we implemented the following three:

- **Computing power**: tasks are sent with higher priority to workers with faster computing speed;

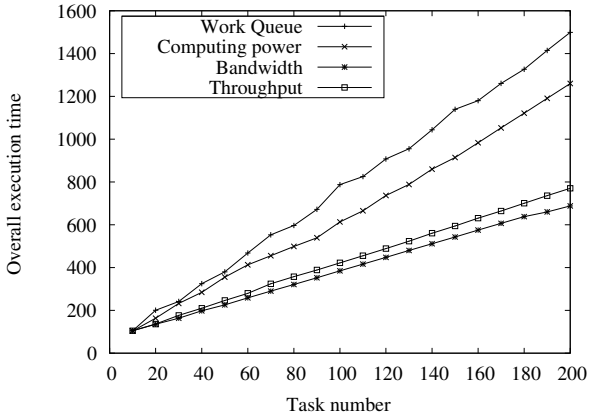

Fig. 2. Comparison of various strategies of independent task allocation.

- **Bandwidth**: tasks are sent with higher priority to workers with faster bandwidth;
- **Throughput**: if c_i (resp. w_i) denotes the time needed by the i-th worker to process (resp. receive data for) a task, then the ratio $w_i/(c_i + w_i)$ represent the throughput for that worker. Tasks are sent with higher priority to workers with higher throughput.

Using SimGrid, it took under two hours to set up the entire simulation framework from scratch with the Work Queue scheduling algorithm and for a platform imported from ENV (shown in Figure 1). Twenty additional minutes were needed to add simulation of bandwidth and processing power measurements for the workers and to use these measurements for implementing the three aforementioned priority-based heuristics. In these simulations, the request data size was set to 100KB and the request computational cost (on the master) was set to 0.1 MFlops. The task data size was set to 10MB and the task computational cost was set to 1 000 MFlops. Network and processor speeds are shown in Figure 1.

A number of experiments can be conducted with the simulator and Figure 2 plots the performance of the four scheduling strategies on the simulated platform in terms of overall execution time versus task number (i.e. the time at which the x^{th} task completes). These simulations ran in less than four minutes on a PIII-733 . The somewhat surprising result is that it is most efficient to prioritize workers according to bandwidth only. In other terms, the master should delegate work as fast as possible. Note that this type of result has been obtained in previous work [2] and that this case study simply confirms this phenomenon on a more realistic platforms.

This case study and other SimGrid examples were used for educational purposes in a course on Parallel Algorithms and Architectures at the École Normale Supérieure de Lyon during the Fall 2002 semester.

VI. Conclusion

In this paper we have recognized two important limitations of simulations used for scheduling research: (i) there is no simulation standard; (ii) simulation methodologies are not appropriate for modern distributed computing platforms such as the Grid. We have presented SimGrid v2 which enables scalable, configurable, extensible, and fast simulations for investigating novel scheduling techniques for these platforms. SimGrid has already been used successfully in [13, 35, 2, 3, 39, 34] and the SimGrid user community is currently undergoing a dramatic expansion. SimGrid is also used for educational purposes in a course on Parallel Algorithms and Architectures at the École Normale Supérieure de Lyon.

Future directions include mechanisms to import Brite [26] topologies and associate them link characteristics in an ENV [32] fashion for SimGrid. We are also creating a repository where SimGrid users can download pre-generated topologies.

A difficult question is that of simulation validation and few projects have addressed it successfully. The rationale is that a simulation implements a model that makes it possible to explore and reason about a systems. Whether the model actually reflects reality is somewhat of a different question. SimGrid implements a number of models that have been widely accepted in the scheduling research community. Nevertheless, we have also designed and implemented models that we believe are more realistic and for which we have conducted initial validation experiments (see Section IV-C).

All details, publications, and software for SimGrid are available at: `http://grail.sdsc.edu/simgrid`.

References

[1] Bajaj, S., Breslau, L., Estrin, D., Fall, K., Floyd, S., Haldar, P., Handley, M., Helmy, A., Heidemann, J., Huang, P., Kumar, S., McCanne, S., Rejaie, R., Sharma, P., Varadhan, K., Xu, Y., Yu, H., and Zappala, D. Improving simulation for network research. Tech. Rep. 99-702, University of Southern California, 1999.

[2] Beaumont, O., Carter, L., Ferrante, J., Legrand, A., and Robert, Y. Bandwidth-centric allocation of independent tasks on heterogeneous platforms. In *International Parallel and Distributed Processing Symposium IPDPS'2002* (2002), IEEE Computer Society Press. Extended version available as LIP Research Report 2001-25.

[3] Beaumont, O., Legrand, A., and Robert, Y. Optimal algorithms for scheduling divisible workloads on heterogeneous systems. Tech. Rep. 2002-36, LIP, Oct. 2002.

[4] Buyya, R., and Murshed, M. GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing. *The Journal of Concurrency and Computation: Practice and Experience (CCPE)* (2002). to appear.

[5] Calvert, K. L., Doar, M. B., and Zegura, E. W. Modeling internet topology. *IEEE Communications Magazine 35*, 6 (June 1997), 160–163.

[6] Casanova, H. Simgrid: A Toolkit for the Simulation of Application Scheduling. In *Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGrid'01)* (May 2001), pp. 430–437.

[7] Casanova, H., and Marchal, L. A network model for simulation of grid application. Tech. Rep. 2002-40, LIP, Oct. 2002.

[8] Chen, Y., Winslett, M., Kuo, S., Cho, Y., Subramaniam, M., and Seamons, K. E. Performance modeling for the Panda array I/O library. In *Proceedings of Supercomputing '96* (1996), ACM Press and IEEE Computer Society Press.

[9] Chiu, D. N. Some observations on fairness of bandwidth sharing. Tech. rep., Sun Microsystems, 1999.

[10] Dickens, P. M., Heidelberger, P., and Nicol, D. M. A distributed memory LAPSE: Parallel simulation of message-passing programs. In *Proceedings of the 8th Workshop on Parallel and Distributed Simulation (PADS '94)* (1994).

[11] Doar, M. A better model for generating test networks. In *Proceedings of Globecom '96* (Nov. 1996).

[12] Entropia. URL: http://www.entropia.com.

[13] Faerman, M., Birnbaum, A., Casanova, H., and Berman, F. Resource Allocation for Steerable Parallel Parameter Searches. In *Proceedings of the Grid Computing Workshop* (Nov. 2002).

[14] Faloutsos, M., Faloutsos, P., and Faloutsos, C. On power-law relationships of the internet topology. In *SIGCOMM* (1999), pp. 251–262.

[15] Fishwick, P. SimPack: Getting Started With Simulation Programming In C And C++. In *Proceedings of the Winter Simulation Conference* (1992), pp. 154–162.

[16] Floyd, S., and Fall, K. Promoting the use of end-to-end congestion control in the Internet. *IEEE/ACM Transactions on Networking 7*, 4 (1999), 458–472.

[17] Foster, I., and Kesselman, C., Eds. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, Inc., San Francisco, USA, 1999.

[18] Gomes, F., Franks, S., Unger, B., Xiao, Z., Cleary, J., and Covington, A. SimKit: A High Performance Logical Process Simulation Class Library in C++. In *Proceedings of the 1995 Winter Simulation Conference* (December 1995), pp. 706–713.

[19] GT-ITM: Georgia Tech Internetwork Topology Models. http://www.cc.gatech.edu/projects/gtitm/.

[20] Howell, F., and R., M. SimJava: A Discrete Event Simulation Package for Java with Applications in Computer Systems Modelling. In *Proceedings of the First International Conference on Web-based Modelling and Simulation* (Jan 1998).

[21] Kielmann, T., Bal, H. E., Maassen, J., van Nieuwpoort, R., Eyraud, L., Hofman, R., and Verstoep, K. Programming environments for high-performance grid computing: the albatross project. *Future Generation Computer Systems* (2002).

[22] Legrand, A., and Lerouge, J. Metasimgrid : Towards realistic scheduling simulation of distributed applications. Tech. Rep. 2002-28, LIP, July 2002.

[23] Liu, J., and Nicol, D. M. *DaSSF 3.1 User's Manual*, Apr. 2001.

[24] Massoulié, L., and Roberts, J. Bandwidth sharing: Objectives and algorithms. In *INFOCOM (3)* (1999), pp. 1395–1403.

[25] Mathis, M., Semke, J., and Mahdavi, J. The macroscopic behavior of the TCP congestion avoidance algorithm. *Computer Communications Review 27*, 3 (1997).

[26] Medina, A., Lakhina, A., Matta, I., and Byers, J. BRITE: Universal topology generation from a user's perspective. Tech. Rep. 2001-003, Computer science department Boston University, 1 2001.

[27] Medina, A., Matta, I., and Byers, J. On the origin of power laws in internet topologies. Tech. Rep. 2000-004, Computer Science Departement, Boston University, 20, 2000.

[28] Miller, A., Nair, R., and Zhang, Z. JSIM: A Java-Based Simulation and Animation Environment. In *Proceedings of the 30th Annual Simulation Symposium (ANSS'97)* (April 1997), pp. 31–42.

[29] The network simulator - ns-2. http://www.isi.edu/nsnam/ns.

[30] Objective modular network testbed. http://www.hit.bme.hu/phd/vargaa/omnetpp.htm.

[31] Schwetman, H. CSIM: A C-based, process oriented simulation language. In *Proceedings of the 1986 Winter Simulation Conference* (December 1986), pp. 387–396.

[32] Shao, G., Berman, F., and Wolski, R. Using effective network views to promote distributed application performance. In *International Conference on Parallel and Distributed Processing Techniques and Applications* (June 1999).

[33] Song, H. J., Liu, X., Jakobsen, D., Bhagwan, R., Zhang, X., Taura, K., and Chien, A. A. The microgrid: a scientific tool for modeling computational grids. In *Supercomputing* (2000).

[34] Su, A., Berman, F., and Casanova, H. Performance Modeling for Entity-level Simulations. In *Proceedings of the Parallel and Distributed Scientific and Engineering Computing with Applications Workshop, Nice, France* (April 2003). to appear.

[35] Su, A., Casanova, H., and Berman, F. Utilizing DAG Scheduling Algorithms for Entity-Level Simulations. In *Proceedings of the 10th Symposium on High Performance Computing 2002 (HPC'02)* (Apr. 2002).

[36] Sugih, J. W. Inet-3.0: Internet topology generator. Tech. Rep. 456-02, Computer science department University of Michigan, 2002.

[37] Takefusa, A., Matsuoka, S., Nakada, H., Aida, K., and Nagashima, U. Overview of a Performance Evaluation System for Global Computing Scheduling Algorithms. In *In Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing (HPDC-8)* (August 1999), pp. 97–104.

[38] Toh, S. SimC: A C Function Library for Discrete Simulation. In *Proceedings of the 11th Workshop in Parallel and Distributed Simulation* (January 1993), pp. 18–20.

[39] Yang, Y., and Casanova, H. UMR: A Multi-Round Algorithm for Scheduling Divisible Workloads. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'03), Nice, France* (April 2003). to appear.