Versatile, Scalable, and Accurate Simulation of Distributed Applications and Platforms

Henri Casanova[a], Arnaud Giersch[b], Arnaud Legrand[c], Martin Quinson[d], Frédéric Suter[e,f,*]

[a] *Dept. of Information and Computer Sciences, University of Hawai'i at Manoa, U.S.A*
[b] *FEMTO-ST, University of Franche-Comté, Belfort, France*
[c] *LIG, CNRS, Grenoble University, France*
[d] *LORIA, Université de Lorraine, France*
[e] *IN2P3 Computing Center, CNRS/IN2P3, Lyon-Villeurbanne, France*
[f] *LIP, INRIA, ENS Lyon, Lyon, France*

Abstract

The study of parallel and distributed applications and platforms, whether in the cluster, grid, peer-to-peer, volunteer, or cloud computing domain, often mandates empirical evaluation of proposed algorithmic and system solutions via *simulation*. Unlike direct experimentation via an application deployment on a real-world testbed, simulation enables fully repeatable and configurable experiments for arbitrary hypothetical scenarios. Two key concerns are accuracy (so that simulation results are scientifically sound) and scalability (so that simulation experiments can be fast and memory-efficient). While the scalability of a simulator is easily measured, the accuracy of many state-of-the-art simulators is largely unknown because they have not been sufficiently validated. In this work we describe recent accuracy and scalability advances made in the context of the SimGrid simulation framework. A design goal of SimGrid is that it should be versatile, i.e., applicable across all aforementioned domains. We present quantitative results that show that SimGrid compares favorably to state-of-the-art domain-specific simulators in terms of scalability, accuracy, or the trade-off between the two. An important implication is that, contrary to popular wisdom, striving for versatility in a simulator is not an impediment but instead is conducive to improving both accuracy and scalability.

Keywords: Simulation, validation, scalability, versatility, SimGrid

1. Introduction

The use of parallel and distributed computing platforms is pervasive in a wide range of contexts and for a wide range of applications. *High Performance Computing* (HPC) has been a consumer of and driver for these platforms. In particular, commodity clusters built from off-the-shelf computers interconnected with switches have been used for applications in virtually all fields of science and engineering, and exascale systems with millions of cores are already envisioned. Platforms that aggregate multiple clusters over wide-area networks, or *grids*, have received a lot of attention over the last decade with both specific software infrastructures and application deployments. Distributed applications and platforms have also come to prominence in the *peer-to-peer* and *volunteer computing* domains (e.g., for content sharing, scientific computing, data storage and retrieval, media streaming), enabled by the impressive capabilities of personal computers and high-speed personal Internet connections. Finally, *cloud computing* relies on the use of large-scale distributed platforms that host virtualized resources leased to consumers of compute cycles and storage space.

While large-scale production platforms have been deployed and used successfully in all these domains, many open questions remain. Relevant challenges include resource management, resource discovery and monitoring, application scheduling, data management, decentralized algorithms, electrical power management, resource economics, fault-tolerance, scalability, and performance. Regardless of the specific context and of the research question at hand, studying and understanding the behavior of applications on distributed platforms is difficult. The goal is to assess the quality of competing algorithmic and system designs with respect to precise objective metrics. Theoretical analysis is typically tractable only when using stringent and ultimately unrealistic assumptions. As a result, relevant research is mostly empirical and proceeds as follows. An *experiment* consists in executing a

*Corresponding author (frederic.suter@cc.in2p3.fr)

software application on a target *hardware platform*. We use the term "application" in a broad sense here, encompassing a parallel scientific simulation, a peer-to-peer file sharing system, a cloud computing brokering system, etc. The application execution on the platform generates a time-stamped trace of events, from which relevant metrics can be computed (e.g., execution time, throughput, power consumption). Finally, research questions are answered by comparing these metrics across multiple experiments.

One can distinguish three classes of experiments. In *in vivo* experiments an actual implementation of the application is executed on a real-world platform. Unfortunately, real-world production platforms may not be available for the purpose of experiments. Even if a testbed platform is available, experiments can only be conducted for (subsets of) the platform configuration at hand, limiting the range of experimental scenarios. Finally, conducting reproducible in vivo experiments often proves difficult due to changing workload and resource conditions. An alternative that obviates these concerns is *in vitro* experiments, i.e., using emulation (e.g., virtual machines, network emulation). A problem with both in vivo and in vitro experiments is that experiments may be prohibitively time consuming. This problem is exacerbated not only by the need to study long-running applications but also by the fact that large numbers of experiments are typically needed to obtain results with reasonable statistical significance. Furthermore, when studying large-scale applications and platforms, commensurate amounts of hardware resources are required. Even if the necessary resources are available, power consumption considerations must be taken into account: using large-scale platforms merely for performance evaluation experiments may be an unacceptable expense and a waste of natural resources. The third approach consists in running (an abstraction of) the application *in silico*, i.e., using simulation. This approach is typically less labor intensive, and often less costly in terms of hardware resources, when compared to in vivo or in vitro experiments. Consequently, it should be no surprise that many published results in the field are obtained in silico.

Two key concerns for simulation are *accuracy* (the ability to run in silico experiments with no or little result bias when compared to their in vivo counterparts) and *scalability* (the ability to run large and/or fast in silico experiments). A simulator relies on one or more simulation models to describe the interaction between the simulated application and the simulated platform. There is a widely acknowledged trade-off between model accuracy and model scalability (e.g., an analytical model based on equations may be less accurate than a complex event-drive procedure but its evaluation would also be less memory- and CPU-intensive). Simulation has been used in some areas of Computer Science for decades, e.g., for microprocessor and network protocol design, but its use in the field of parallel and distributed computing is less developed. While the scalability of a simulator can be easily quantified, evaluating its accuracy is painstaking and time-consuming. As a result, published validation results often focus on a few scenarios, which may be relevant to a particular scope, instead of engaging in a systematic and critical evaluation methodology. Consequently, countless published research results are obtained with simulation methods whose accuracy is more or less unknown.

An important observation is that simulators used by parallel and distributed computing researchers are domain-specific (e.g., peer-to-peer simulators, grid simulators, HPC simulators). In some cases, domain-specificity is justified. For instance, wireless networks are markedly different from wired networks and in this work, for instance, we only consider wired networks. But, in general, many simulators are developed by researchers for their own research projects and these researchers are domain experts, not simulation experts. The popular wisdom seems to be that developing a versatile simulator that applies across domains is not a worthwhile endeavor because specialization allows for "better" simulation, i.e., simulations that achieve a desirable trade-off between accuracy and scalability. In this work, we *rebut popular wisdom* and claim that, when developing a simulation framework, *aiming for versatility is the way to achieve better accuracy and better scalability*. Our main contribution is that we confirm this claim by synthesizing the experience gained during the 10-year development of the SimGrid discrete-event simulation framework, presenting results relating to both simulation design and simulation implementation. Some of these results have been previously published in conference proceedings, as referenced hereafter, while others are novel contributions.

The rest of this article is organized as follows. Section 2 presents related work. Section 3 discusses the current design and design goals of SimGrid. Sections 4 and 5 explain how striving for versatility has led to advances in accuracy and scalability, respectively. While these sections include several short case studies, Section 6 presents a full-fledged case study in the HPC domain. Finally, Section 7 concludes the paper with a brief summary of findings and with perspectives on future work.

2. Related Work

In this section we discuss popular simulators that have been used in the last decade and whose goal is to enable "fast" simulation of grid, cloud, peer-to-peer, volunteer, or HPC applications and platforms, meaning that the simulation time (i.e., the runtime of the in silico experiment) should be orders of magnitude faster than the

Table 1: State-of-the-art simulators from various communities and modeling approaches.

Simulator	Community of Origin					Application			Network						CPU		Disk		
	high performance computing	grid computing	cloud computing	volunteer computing	peer-to-peer computing	execution trace	abstract specification	programmatic specification	latency	bandwidth	store-and-forward	ad hoc flow	TCP flow-level model	packet-level	scaled measured delay	scaled user-provided delay	capacity	seek + transfer	block-level
PSINS [1]	✓					✓			✓	✓		✓			✓	✓			
LogGOPSim [2]	✓					✓			✓	✓						✓			
BigSim [3]	✓					✓			✓	✓				✓	✓	✓			
MPI-SIM [4]	✓							✓	✓	✓					✓	✓			
OptorSim [5]		✓						✓	✓	✓		✓				✓	✓		
GridSim [6]		✓						✓	✓	✓	✓	✓				✓	✓	✓	
GroudSim [7]		✓	✓					✓	✓	✓		✓				✓			
CloudSim [8]			✓					✓	✓	✓	✓					✓	✓	✓	
iCanCloud [9]			✓					✓	✓	✓				✓		✓	✓	✓	✓
SimBA [10]				✓			✓	✓	✓							✓			
EmBOINC [11]				✓			✓	✓	✓							✓			
SimBOINC [12]				✓				✓	✓	✓			✓			✓			
PeerSim [13]					✓		✓		✓										
OverSim [14]					✓			✓	✓	✓				✓					
SimGrid [15]		✓				✓	✓	✓	✓	✓			✓	✓	✓	✓	✓	✓	

simulated time (i.e., the simulated runtime of the application). Most of these simulators share the same design with three components: (i) simulation models; (ii) platform specification; and (iii) application specification. *Simulation models* are used to implement the evolution of simulated application activities (computations, data transfers) that use simulated resources (compute devices, network elements, storage devices) throughout simulated time. More specifically, given all the application activities that use a set of resources, resource models are used to compute the completion date of the activity that completes the earliest and the progress made by all other activities by that date. *Platform specification* mechanisms allow users to instantiate platform scenarios without having to modify the simulation models or the simulation's implementation. Each resource must be described using an instantiated simulation model, and resources can be connected together (e.g., a particular set of network links and routers is used for end-to-end communication between two compute resources). *Application specification* refers to the set of mechanisms and abstractions for users to describe the nature and sequence of activities that must be simulated. Existing simulators provide many options for application specification ranging from abstract finite automata to actual application implementations.

In Section 2.1, we introduce the most prominent and relevant simulation frameworks. We then discuss the design choices in these simulators, and the rationales for these choices, for each of the three components above in Sections 2.2, 2.3, and 2.4.

2.1. Related simulators

The simulation of parallel and distributed applications has received a fair amount of attention in the literature. Many simulators have been developed that employ a wide range of simulation techniques. Table 1 summarizes the key features of prominent simulators most relevant to this work. The content of each row is based on the content of published research articles as well as on our own source code inspection whenever available. While some of these simulators are actively used, others seem to be more or less inactive at the time this article is being written. For each simulator we indicate the research community from which it has emerged, the way in which it models the simulated application, and the type of models it uses to simulate network, compute, and storage resources. Details on the particular models are given throughout the rest of this article.

The simulation of parallel applications on parallel computing platforms has a long history in HPC, in particular in the context of applications based on MPI (Message Passing Interface) [16]). Two main approaches are used: *off-line* and *on-line* simulation. In off-line simulation, a time-stamped log of computation and communication events is first obtained by running the application on a real platform. The simulator then replays this sequence

of events as if they were occurring on another platform with different hardware characteristics. We list the three most representative such simulators in Table 1 [1, 2, 3] but several others are available (e.g., [17, 18]). One issue with off-line simulation is that event logs are tied to a particular application execution (e.g., number of processors, block size, data distribution schemes) so that a new log must be obtained for each simulation scenario. However, extrapolation is feasible as proposed for instance in LogGOPSim [2]. The alternative to off-line simulation is on-line simulation in which actual application code is executed on a host platform that attempts to mimic the behavior of the target platform. Part of the instruction stream is intercepted and passed to a simulator. Many on-line simulators have been developed with various features and capabilities (e.g., [4, 19, 20, 21]). In these simulators, the amounts of hardware resources required to run the simulated application on the host platform are commensurate to (or in fact larger than) those needed to run the actual application on the target platform. Simulation scalability is thus achieved by "throwing more hardware" at the problem. In this work we limit our scope to simulations that can be executed on a single computer, so that simulation scalability must be achieved in software. Due to this fundamental difference between SimGrid and on-line HPC simulators, we only show one such simulator in Table 1 (MPI-SIM [4]).

Simulators have also been developed in the area of *grid computing*, most of which only intended for use by their own developers. Some were made available to the community but proved short-lived, such as OptorSim [5] (shown in the table) or ChicSim [22]. Besides SimGrid, the other simulator widely used in grid computing research is GridSim [6]. More recently, simulators have been proposed for simulating *cloud computing* platforms and applications. GroudSim [7] is a framework that enables the simulation of both grid and cloud systems. CloudSim [8] builds on the same simulation internals as GridSim but exposes specific interfaces for simulating systems that support cloud services. iCanCloud [9] has been specifically developed to simulate cloud platforms and applications.

Simulators have also been developed for simulating *volunteer computing* systems, i.e., systems that consist of large numbers of individually owned and volatile hosts. BOINC [23] is the most popular volunteer computing infrastructure today, and these simulators attempt to simulate (parts of) BOINC's functionalities. In fact, BOINC itself embeds in its source code a simple time-driven simulator for running the actual client scheduler code in simulation mode. The SimBA simulator [10] models BOINC clients as finite-state automata based on probabilistic models of availability, and makes it possible to study server-side scheduling policies in simulation. The same authors later developed EmBOINC [11]. Unlike SimBA, EmBOINC executes actual BOINC production code to emulate the BOINC server. SimBOINC [12] goes further and simulates the full BOINC system by linking the BOINC code with SimGrid, thus allowing for multiple servers and for the simulation of client-side scheduling.

Another area in which simulators have been developed is *peer-to-peer computing* [24]. Most of these simulators trade off accuracy for scalability, so as to make it possible to simulate up to millions of peers. For instance, it is common to simulate network transfers as fixed delays since message count is a useful metric to evaluate peer-to-peer systems. PeerSim [13] is likely the most widely used simulators for theoretical peer-to-peer studies, and relies on simplistic but scalable simulation models. OverSim [14] relies on the OMNeT++[25] discrete-event simulation kernel for implementing more realistic packet-level network simulation. Several other simulators have emerged, such as P2PSim [26] or PlanetSim [27], which have been short-lived and are no longer maintained. It is thus difficult to say whether more recent proposals, e.g., D-P2P-Sim [28], will perdure.

2.2. Simulation models

Many simulation models for compute, network, and storage resources have been proposed in the literature, ranging from simple mathematical equations to complex processes. For instance, for a hard drive, access time could be modeled as a seek time plus the data size divided by a bandwidth, or instead emerge from a detailed discrete-event simulation that accounts for platters, sectors, blocks, hardware/software buffers, file system overheads, etc. At one extreme the first model would be very scalable but likely inaccurate, while at the other extreme the second model would be unscalable but (hopefully) accurate. In general, different models achieve different trade-offs between the time it takes to evaluate them and the level of detail with which they capture the behavior of physical resources.

In what follows we discuss the simulation models implemented as part of state-of-the-art simulators of parallel and distributed computing applications. For CPU and storage models, there is a strong consensus among the simulators: the vast majority are at one extreme (simplistic analytical models) for both types of resources. By contrast, the design space is much larger for network resource models and we see more diversity in state-of-the-art simulators. This larger design space provides more of an opportunity to seek models that achieve judicious trade-offs between accuracy and scalability (as seen in Section 4). This is why the discussions of CPU and storage resource models (Sections 2.2.1 and 2.2.2) are much shorter than the discussion of network models (Section 2.2.3).

2.2.1. CPU models

The high-accuracy, low-scalability extreme for simulating CPUs is cycle-level (also called cycle-accurate) simulation, which has been used occasionally in the HPC context [21] but precludes fast simulations due to CPU-intensiveness. Furthermore, it is labor-intensive to instantiate a cycle-level simulator to simulate a precise architecture. To complicate matters, it has been shown that in some instances cycle-level simulators do not necessarily lead to accurate results, or at least not as accurate as one might expect [29]. For these reasons, the common approach used by all simulators in Table 1 is to employ a simple analytical model of compute delay. More specifically, task execution times are computed by dividing a compute cost (e.g., number of instructions) by a compute speed (e.g., number of instructions per time unit), with possibly a random component. The compute speed can be instantiated for various simulated resources based on benchmark results obtained on corresponding real-world resources. This model can lead to reasonable results for simple CPU-bound computation and can even be used to model simple multi-core parallelization of computation. But, in general, it is limited because it does not account for architecture-specific features of the simulated compute resource (memory hierarchy, CPU architecture, GPU architecture, on-chip buses and networks, etc.). While one could envision more sophisticated analytical models that capture some of these features without resorting to cycle-level simulation, designing such models is an open research question.

2.2.2. Storage models

It may seem surprising that only a few simulators in Table 1 provide a notion of simulated storage resources. We hypothesize that the reason why storage resource simulation is rarely done is twofold. First, not all users require simulation of storage resources and it is assumed that those who do can implement their own storage simulation models. Second, accurate modeling of storage resources such as hard drives and solid-state drives is known to be extremely challenging. The high-accurate, low-scalability option is the discrete-event simulation of storage resources as for instance done in the DiskSim simulator [30]. This simulator models the operation of the storage hardware precisely and could serve as a basis for implementing a storage system simulator that models other hardware components (e.g., buses and networks) and software components (e.g., file systems). This is the approach used in [31], for instance, which targets fine-grain discrete-event simulation of storage area networks. However, this approach is rarely used when simulating parallel and distributed applications due to long simulation times and because correctly instantiating such complex models is difficult.

A few simulators from the grid computing and cloud computing domains provide simple storage access time models. For instance GridSim, CloudSim, and SimGrid model data access times using a simple model based on a (fixed or randomly generated) seek time and a fixed data transfer rate. This model is not truly representative of real storage resources since caching, locality, and file system effects are not captured. These effects are known to be performance drivers but are also known to make the accurate analytical modeling of storage systems an open question. Among the simulators in Table 1, iCanCloud provides the most sophisticated model: it considers individual disk blocks and simulates seek times based on the locality of block accesses. Instantiating such a model in a realistic way is, however, non-trivial.

2.2.3. Network models

All simulators of parallel and distributed applications implement some model for the simulated platform's network. The simulators listed in Table 1 are diverse in their approach to network modeling. One network simulation approach that is acknowledged to have high accuracy because it captures most real-world phenomena is packet-level simulation. Packet-level simulators implement full-fledged network protocols and are used extensively for network research (e.g., the ns-3 simulator [32]). The issue with packet-level simulation in our context, that is the simulation of large-scale and perhaps long-running parallel and distributed applications, is the lack of scalability due to long simulation times, which can be orders of magnitude larger than simulated time [33]. As a result, packet-level simulation is not usable for typical grid/cloud computing simulations (some authors have used it for simulating HPC applications on clusters [34] but with long simulation times). Some of the simulators in Table 1 provide packet-level simulation as an option. For instance SimGrid does provide an interface to ns-3. Users who can tolerate long simulation times may then benefit from more accurate network simulations when needed. But the vast majority of users need fast simulations, which can only be achieved by using analytical, and thus potentially less accurate, network models.

Some simulators implement analytical models that by design ignore network phenomena that are deemed irrelevant to the target simulation domain. For instance, PeerSim ignores bandwidth effects because it is designed for simulations with many small messages and for users who care about message counts more than about data transfer rates. Likewise, volunteer computing simulators such as SimBA or EmBOINC only model network latencies. GroudSim does not model network contention that may happen in the core of the network and simply

assumes that each host is bandwidth-limited by its own Internet connection. Network contention is also ignored by most HPC simulators such as LogGOPSim, BigSim, or MPI-SIM. The LogGOPSim authors simply state that the platforms they target have sufficient network provisioning so that contention does not occur. Such design choices severely limit the versatility of the simulator, but these simulators are admittedly not designed for versatility and they can claim accuracy for those scenarios for which they were designed.

The accuracy of a simulator can be evaluated by confronting simulation results to a ground truth. While experiments on a real network can provide this ground truth, these experiments are typically limited to a few network configurations. Instead, a more feasible approach is to use results obtained with a packet-level simulator as the ground truth. Many published works that propose a simulator include a section devoted to evaluating the accuracy of the network model. Unfortunately, most of these evaluations are either merely qualitative or consist in exhibiting a few "good cases" in which simulation results lead to reasonable quantitative trends. Few direct comparisons to packet-level simulations or real executions are actually attempted.

Given the above, it is fair to say that the majority of simulation results published in the area of parallel and distributed computing are of unknown and thus questionable validity. Even using popular simulators, it is often straightforward to construct simple and relevant use cases for which plainly invalid results are obtained [35]. Some authors are explicit about the validity limitations of their simulators. For instance, the authors of GroudSim acknowledge in [7] that their bandwidth sharing model is flawed when competing flows have heterogeneous bottleneck bandwidth constraints, which unfortunately is a common case in real-world networks. Similarly, the authors of OptorSim document in [5] that in their network model the bandwidth share that each flow receives on a congested network link does not take into account the fact that some of these flows may be limited by other links in their paths. For both these simulators the implication is a waste of available network bandwidth when compared to real networks but, at least, the authors provide a sense of how much trust should be put into simulation results.

In many cases the validity limits of a simulator are undocumented. This is the case for the GridSim [6] grid simulator and its follow-up cloud computing simulator, CloudSim [8]. These simulators are extremely popular in their communities (e.g., the GridSim distribution has been downloaded 20,000 times since 2007 or over 10 times a day). These simulators have thus been the basis for hundreds of published articles as well as for other recent simulators [36, 37, 38, 39]. CloudSim builds on GridSim to provide a simulator for cloud computing. In [8] the authors state the following rationale: "Since SimJava and GridSim have been extensively utilized in conducting cutting edge research in Grid resource management by several researchers, bugs that may compromise the validity of the simulation have been already detected and fixed." Unfortunately, this claim is simply unrealistic given how rarely true validation studies are attempted. And indeed, a quick inspection of both GridSim's and CloudSim's code suggests very simple invalidating cases [35].

2.3. Platform specification

Once models of resources have been chosen, it is necessary to (i) instantiate each resource model with appropriate parameters; and (ii) describe the interconnections of the resources.

The instantiation of resource models for individual resources is done by specifying a few parameters. For the simulators in Table 1, the CPU resource model takes one parameter (the computation rate, in FLOPS or MIPS), the storage resource model takes up to two parameters (seek time and I/O bandwidth), and the network resource model takes up to two parameters (link latency and link bandwidth). These parameters can be chosen as constants or sampled from relevant probability distributions. These model instantiations can be provided by users either via text description files or via a programmatic interface. A text interface offers a clear separation between the simulated application and the specification of the simulated execution environment. A programmatic interface provides increased expressiveness power since repetitive patterns can get generated from compact programmed descriptions.

Given a set of resources, each instantiated with a model, a platform description must list all network-reachable elements (hosts, routers, links) and the topological interconnections of these elements, i.e., allowed network paths. Most simulators allow to interconnect links and hosts by expressing one-hop routes, and then route messages using the shortest path on the topology graph. This approach is scalable because shortest paths can be computed in polynomial time and one only needs to store the topological graph. However, it may be inaccurate because in real networks routing exhibits irregularities (e.g., asymmetric paths). A more accurate solution is to specify explicit routing tables, making it possible to describe more realistic networks, thus placing a higher burden on the user. This approach is less scalable because routing tables must be stored and routes computed from these tables, leading to potentially large memory and CPU requirements for simulating large-scale networks. As discussed in Section 5.2 it is possible to exploit repetitive patterns so as to allow for platform descriptions that are both accurate and scalable.

2.4. Applications specification

Given a fully specified platform, one must express how its resources are used by the simulated application. There are three main approaches: (i) off-line simulation; (ii) formal description; and (iii) programmatic description.

Off-line simulation consists in replaying event traces captured during the execution of the application on a real platform. This approach is commonplace in the HPC community for simulating the execution of MPI applications. It may not be scalable as event traces can be large, but it describes the execution of the application accurately since the trace is generated from a real execution. A drawback is that in many cases researchers do not have access to an actual application implementation, and in fact they may want to explore various application design schemes in simulation before committing to developing such an implementation.

The second approach does not require an application implementation, but instead consists in developing a formal description of the application execution, e.g., as finite automata. This is the approach used for instance by PeerSim and SimBA. These particular simulators opt for a formal description because it is compact and thus affords scalability. However, this description can be too constraining since complex application logic may be too difficult to describe within such a rigid formalism. In this case the formal description may be only an approximation of the actual application to be simulated.

Most simulators follow the third approach, by which users describe simulated applications programmatically as sets of functions/methods that describe Concurrent Sequential Processes (CSP). The description is thus still scalable because compact, and is more accurate than formal descriptions based on automata. For this reason, PeerSim provides such programmatic description as an alternative to automata. Once programmatically described, the simulated application can then be executed by virtualizing each simulated process into one execution context. Several technologies can be used in this view, the most natural approach being the encapsulation of each simulated process into a thread, as done in GridSim for instance. The use of threads suffers from scalability limitations. These limitations can be alleviated by using continuations instead of threads (see Section 5.3).

3. SimGrid design and objectives

3.1. Software stack

Figure 1 shows the main components in the design of SimGrid and depicts some of the key concepts in this design. The top part of the figure shows the three APIs through which users can develop simulators. The MSG API allows users to describe a simulated application as a set of concurrent processes. These processes execute code implemented by the user (in C, C++, Java, Lua, or Ruby), and place MSG calls to simulate computation and communication activities. The SMPI API is also used to simulate applications as sets of concurrent processes, but these processes are created automatically from an existing application written in C or Fortran that uses the MPI standard. SMPI also includes a runtime system, not shown in the figure, that implements necessary MPI-specific functionalities (e.g., process startup, collective communications). MSG thus makes it possible to simulate any arbitrary application, while SMPI makes it possible to simulate existing, unmodified MPI applications. The mechanisms for simulating the concurrent processes for both MSG and SMPI are implemented as part of a layer called SIMIX, which is a kernel (in the Operating Systems sense of the term) that provides process control and synchronization abstractions. The set of concurrent processes is depicted in the SIMIX box in the figure. All processes synchronize on a set of condition variables, also shown in the figure. Each condition variable corresponds to a simulated activity, computation or data transfer, and is used to ensure that concurrent processes wait on activity completions to make progress throughout (simulated) time. The third API, SimDAG, does not use concurrent processes but instead allows users to specify an abstract task graph of communicating computational tasks with non-cyclic dependencies.

Regardless of the API used, the simulation application consists of a set of communication and computation activities which are to be executed on simulated hardware resources. Compute resources are defined in terms of compute capacities (e.g., CPU cycles per time unit). They are interconnected via a network topology that comprises network links and routing elements, defined by bandwidth capacities and latencies. All resources can be optionally associated with time-stamped traces of available capacity values including possible downtime. An example specification of available resources is depicted in the bottom-right of Figure 1, highlighting three network links (L_1, L_2, L_m) and one compute resource (P_1).

The simulation core, i.e., the component that simulates the execution of activities on resources, is called SURF and is shown in the bottom-left of the figure. Each activity is defined by a total amount of work to accomplish (e.g., number of CPU cycles to execute, number of bytes to transfer) and a remaining amount of work. When its remaining amount of work reaches zero the activity completes, signaling the corresponding SIMIX condition variable or resolving a task dependency in SimDAG. Activity i corresponds to a variable, x_i, which represents a

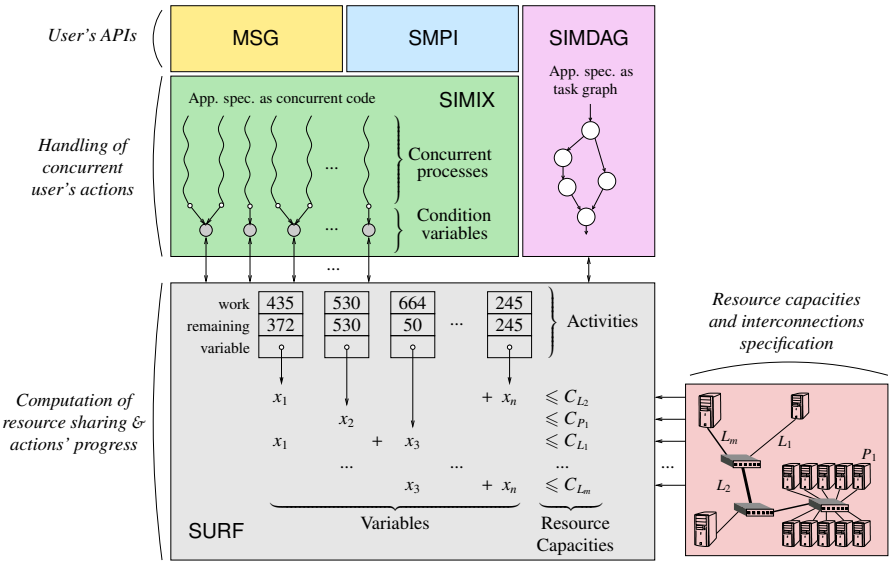


Figure 1: Design and internals of SimGrid.

resource share used by the activity. A set of constraints over these variables, with one constraint per simulated resource, then describes how the activities compete for using the resources. An example is shown in the figure. The right-hand sides of each constraint is a resource capacity, denoted by C_x where x is a given resource (e.g., C_{L_2} is the capacity of network link L_2). The first activity requires performing 435 units of work, 372 of which remains to be performed at the current simulated date, and is associated to variable x_1. Variable x_1 participates in two constraints, for resources L_2 and L_1, meaning that in this example the first activity is a data transfer that uses network links L_2 and L_1, among others. The third activity is also a data transfer, with 50 units of work remaining out of 664. This transfer uses links L_1 and L_m, meaning that it shares the bandwidth capacity of L_1 with the first activity (as seen in the $x_1 + x_3 \leqslant C_{L_1}$ constraint). The n-th activity uses links L_2 and L_m, and its corresponding variable x_n thus appears in those two constraints. Finally, the second activity in this example corresponds to a computation and its variable x_2 appears in the constraint $x_2 \leqslant C_{P_1}$, showing that this resource is not shared with any other compute activity. Note that the second and n-th activities in this example have yet to begin as their remaining works are equal to their total works. Based on these constraints the simulation core computes resource allocations so as to optimize a relevant objective function, as explained in upcoming sections.

3.2. Accuracy, scalability, and versatility

We have seen that the two primary concerns of the users of a simulator are *accuracy* and *scalability*. These two concerns typically conflict and the simulators mentioned in Section 2 often explicitly trade off one for the other. Nevertheless, throughout the history of the SimGrid project we have striven to improve both accuracy and scalability. The primary motivation has been to achieve a third objective, *versatility*. It is important to distinguish versatility from genericity. A selling point of many simulators is that their design is generic. This is arguably always desirable as it makes it possible to augment/replace functionality within the same overall software design (e.g., the network model is accessed via a well-defined interface so that new models can be implemented and integrated). SimGrid does provide some genericity, which we view simply as good software engineering. By contrast, versatility implies that the simulator's implementation provides the necessary capabilities to run simulations for multiple domains accurately and scalably. In this sense, many simulators discussed in the previous section are generic but not versatile.

A simulator that aims for versatility must provide simulation models that subsume and improve on models used by domain-specific simulators; it must allow for the description of arbitrary distributed applications to cover a spectrum of domains (e.g., from HPC to peer-to-peer computing); and it must allow for the description of arbitrary simulated platforms in which resources can be described by a range of instantiated models. To provide such capabilities, a simulator should not only provide different simulation model implementations but should also be designed with versatility in mind, which has led us to use the design shown in Figure 1. The key aspect of this design, which in hindsight may seem natural but is not necessarily used by the simulators reviewed in Section 2, is the complete separation between simulated resource specification, simulated application execution, and resource sharing models.

SimGrid, at least its most recent versions, has achieved versatility and is used in domains including cluster, grid, cloud, volunteer, and peer-to-peer computing, as well as various other distributed computing settings. The natural expectation was that aiming for versatility would have detrimental effects on accuracy and scalability, or at least on the trade-off between the two. *Our main observation is that, instead, striving for versatility has been key for improving both accuracy and scalability*. In fact, we contend that several of the improvements we have achieved would not have been possible without considering versatility as a primary objective. As a side-effect, increased versatility has provided a stronger motivation to improve accuracy and scalability, since improvements translate to multiple simulation domains and thus to a larger user community. The next two sections describe several accuracy and scalability advances made in SimGrid, emphasizing the role of the versatility design objective.

4. Versatile yet accurate simulations

SimGrid uses a unified model for simulating the execution of activities on simulated resources. This model is purely analytical so as to afford scalability by avoiding cycle-, block-, and packet-level simulation of compute, storage, and network resource usage. Formally, given a resource r, and a set of simulated activities, $\mathcal{A}$, the model specifies the following constrained Max-Min optimization problem:

$$\begin{array}{l} \text{MAXIMIZE } \min_{a \in \mathcal{A}} \ \varrho_a \\ \text{UNDER CONSTRAINTS} \\ \left\{ \sum_{a \in \mathcal{A} \text{ using resource } r} \ \varrho_a \leqslant C_r, \right. \end{array} \tag{1}$$

where C_r denotes the capacity of resource r, and ϱ_a denotes the resource share allocated to activity a. Solving this optimization problem, which boils down to solving a linear system, yields instantaneous resource shares given which resources are used by which activities. Given these computed resource shares at simulated time t_0, for all simulated resources, the SURF component of SimGrid computes the first activity that will complete, advances the simulated clock to that time, say t_1, removes the completed activity from consideration, accounts for the progress of each activity given its resource shares and the simulated elapsed time $t_1 - t_0$, and possibly adds newly created activities.

The optimization problem in Eq. (1) is at the core of the SURF component of SimGrid (see Figure 1), which implements efficient algorithms and data structures to solve the corresponding linear system quickly (see Section 5.1). The key aspect of this model is that it is general and can be used to simulate CPU, storage, and network resources.

Regarding CPU resources, we have seen that relevant simulators in Table 1 all use a simple analytical model by which the CPU is shared fairly among concurrent simulated compute activities. Fair sharing is subsumed by the optimization problem in Eq. (1) (maximizing the minimum of n resource shares that all sum to C_r leads to all shares equal to C_r/n). SimGrid, like other simulators, also allows the notion of compute priorities, so that resource shares are scaled by normalized priorities in Eq. (1).

The analytical models for storage resources used by the simulators in Table 1 (but for iCanCloud, which uses block-level simulation) use fair sharing of disk bandwidth, with optionally an extra fixed seek time. Like for CPU resources, the optimization problem in Eq. (1) can be used to simulate fair sharing of disk bandwidth. The seek time is added as a fixed initial delay when advancing the simulation clock.

For both CPU and storage resources, SimGrid thus complies with the analytical simulation models used by state-of-the-art simulators. These models are simplistic (see Sections 2.2.1 and 2.2.2) but developing more accurate models is an open question, which we do not address in this work. By contrast, there is a clear opportunity when simulating network resources. In Section 2.2.3 we have seen that state-of-the-art analytical network simulation models used by the simulators in Table 1 can lead to documented or undocumented invalid behaviors, even for simple simulation scenarios. It is thus a fair question to ask whether accurate analytical network models are even feasible, and if not then one is left with unscalable packet-level simulation. It turns out that the level of detail provided by packet-level simulation is not necessary for studying large-scale applications that exchange large amounts of data. This provides an opportunity to develop an analytical network model that is scalable and accurate, within reasonable and identifiable limits, thereby bridging the accuracy gap between inaccurate scalable analytical models and accurate unscalable packet-level models. In what follows we discuss how this opportunity is seized in SimGrid.

4.1. An empirically informed model of TCP for moderate size grids

Analytical "flow-level" models have been proposed in the networking literature, mostly to study the theoretical behavior of TCP protocols. Inspired be these developments, in SimGrid we use a flow-level model for the purpose

of network simulation. In a flow-level network model the individual packets of an end-to-end communication are abstracted into a single entity, a *flow*, which is characterized by a data transfer rate, or bandwidth. This bandwidth depends on the network topology and on the interactions with other ongoing network flows. It is assumed that the flows have reached *steady-state*, and the goal is to define analytical bandwidth sharing models that capture the bandwidth sharing behaviors of actual network protocols. In the context of SimGrid, we have focused on TCP as it is ubiquitous in grids and wide area networks, but also in clusters. Eq. (1) corresponds to a a popular bandwidth sharing model, Max-Min fairness [40], by which the bandwidth allocation is such that increasing the allocation of any flow would necessarily require decreasing the allocation of a less favored flow. A single network flow can use multiple network links, and the flow is allocated the same bandwidth on all the links it traverses. As a result, we can write a network-specific version of Eq. (1) as:

$$\begin{array}{l} \textsc{Maximize } \min_{f \in \mathcal{F}} \ \varrho_f, \\ \textsc{under constraints} \\ \left\{ \forall l \in \mathcal{L}, \sum_{f \in \mathcal{F} \text{ going through } l} \varrho_f \leqslant B_l, \right. \end{array} \tag{2}$$

where $\mathcal{L}$ is the set of all network links, $l \in L$ denotes a network link with bandwidth capacity B_l, $\mathcal{F}$ is the set of all simulated network flows, and f denotes a flow with assigned bandwidth ϱ_f.

The above model is simple to implement, has low computational complexity, and as a result is implemented in several simulators [3, 5, 7, 41]. Unfortunately, it is known that TCP does not implement Max-Min fairness [42]. As a result different and more sophisticated bandwidth sharing models have been proposed [43]. In [35], we have described how we improved on standard Max-Min fairness in several ways via the use of additional parameters, and we have shown that this model compares favorably to the models in [43]. These improvements were achieved thanks to a thorough invalidation study that highlighted key characteristics of TCP that are not captured by the simple Max-Min model. Most published simulator evaluation studies focus on demonstrating "good" results for particular cases. Instead we followed the critical method [44], which places model invalidation at the center of the scientific endeavor, thus striving to exhibit "bad" cases so as to understand and hopefully extend the validity limits of our simulation models.

The first weakness of the formulation in Eq. (2) is that it does not account for TCP's flow control mechanism, which is known to prevent full bandwidth usage as flows may be limited by large latencies [45, 46, 47]. Therefore, the transmission rate ϱ_f of a flow f should be bounded by $W_{\max}/RTT_f$ where $W_{\max}$ is the configured maximum congestion window size and RTT_f is the round trip time (RTT) of the packets in the flow. Additionally, the sophisticated Adaptive Increase Multiplicative Decrease congestion window mechanism of TCP leads to RTT-unfairness [48]. In versions of TCP like Reno, two flows contending on the same bottleneck link receive bandwidth shares inversely proportional to their RTTs. This behavior can be captured by modifying the constraints in Eq. (2). Finally, flow throughput can be dramatically affected by reverse-traffic [49]. Such a phenomenon is generally very poorly captured by flow-level models [50, 51, 52]. Yet, simple modifications of the constraints in Eq. (2) make it possible to capture throughput degradation due to reverse traffic, at least locally. All these improvements are described in detail in [35] and precursor articles referenced therein.

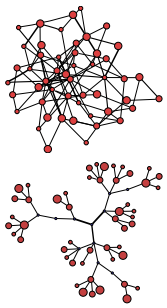
The improved model provides more accurate bandwidth shares, but simulating a flow requires a model of the flow's execution time given its bandwidth share. The common approach is to model the execution time of a flow that transfers S bytes of data as the latency plus S divided by the bandwidth:

$$T_f(S) = \ell_f + S/\varrho_f, \tag{3}$$

where ϱ_f is the bandwidth share computed by the bandwidth sharing model and ℓ_f is the end-to-end latency. This model was shown to lead to good result only for large data sizes on the order of 10 MiB [33]. This is because TCP's slow-start behavior is not captured. While there is no hope for a flow-level model to capture this behavior perfectly, a more accurate empirical model can be derived:

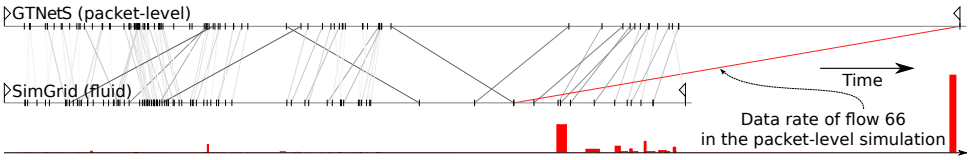
$$T^{\text{improved}} = \alpha\ell_f + \frac{S}{\beta\varrho_f}, \tag{4}$$

where α and β (typically $\alpha \in [10, 15]$ and $\beta \in [0.8, 1]$, depending on the version of TCP [53, 35]) are two additional positive real parameters. Packet-level simulations are used to calibrate parameter values, i.e., to determine the parameter values that minimize modeling error for a set of synthetic simulation scenarios. We found the model to be accurate for data sizes as low as 100 KiB, i.e., about two orders of magnitude smaller than previously achieved. For smaller data sizes the flow-level model leads to transfer times that are too short. Below this limit the assumption that the transfer time is a linear function of the data size breaks down because data is exchanged as discrete network packets. Users wanting to simulate small-size data transfers over wide area networks have



(a) Typical platform graphs used in the study. One hundred 100 MiB transfers between random pairs of endpoints.

(b) Typical situation: the SimGrid flow-level model achieves an excellent prediction when compared to a packet level simulation.



(c) The most extreme situation we stumbled upon where the flow-level model significantly differs from a packet-level model for several flows. The lower part depicts the evolution of the throughput of the last finishing flow in the packet-level simulation and for which the flow-level model has a large error. Because of high contention, this flow stalls 65% of the time. When there is no other remaining connection in the network, it does not transmit a single byte for 380 seconds, and finally completes the transfer in a few seconds.

Figure 2: Comparison of flow completion times with flow-level simulation (top timeline) and packet-level simulation (bottom timeline); lines connecting timeline markers correspond to the same flow in both simulations and are darker for larger completion time mismatches.

two options: either configure SimGrid to use the ns-3 packet-level simulator or account for optimistic simulated transfer times when drawing conclusions from simulation results obtained with the above model.

Overall, results in [53, 35] show that the flow-level model, once improved with the above additional parameters, can capture key characteristics at the macroscopic level of TCP, allows to account for slow start (to some extent), protocol overhead, RTT-unfairness, reverse-traffic, and flow-control limitation, and leads to better results than the simplified model used in earlier versions of SimGrid [54]. This simplified model had already led to drastic improvements compared to the simulators discussed in Section 2.2.3 but its limitations motivated the development of the improved model [33].

As an illustration, Figure 2 compares typical outcomes of invalidation studies presented in [35]. One hundred 100 MiB transfers are launched between random pairs of endpoints for several dozens of randomly generated network topologies such as the ones depicted in Figure 2(a). With the improved model, most scenarios lead to good accuracy, as seen for instance in Figure 2(b). Only a few situations remain, as that shown in Figure 2(c), where we seem to reach the limits of the flow-level approximation. These situations occur for highly contended scenarios (i.e., with extremely small latencies and low bandwidth capacities). In these scenarios the high error is due to the discrete nature of the TCP protocol, which, by design, is not captured by the flow-level approximation.

Our flow-level network model was originally designed for the grid computing domain. However, with the above improvements it is applicable to other scenarios (e.g., under-provisioned networks, applications that exchange as few as a few hundreds of KiB of data), thereby making SimGrid more versatile.

4.2. Extending the model for HPC simulations

The improved model described in the previous section expands the versatility of the simulator toward a broader range of wide-area networks, provided the simulated applications exchange messages on the order of a few hundred KiB. At the other end of the spectrum, many users wish to simulate cluster platforms that consist of compute nodes connected via (a hierarchy of) switches. The goal is to simulate a single cluster, or to simulate intra-cluster phenomena in a grid or cloud platform. In these settings the communication workload often comprises many small messages that consist of a few KiB or even only a few bytes.

Our improved model fails to capture some fundamental aspects of cluster interconnects with TCP and popular MPI implementations, e.g., OpenMPI [55] or MPICH2 [56], over Gigabit Ethernet switches. For instance, a message under 1 KiB fits within an IP frame, in which case the achieved data transfer rate is higher than for larger messages although latency is generally also larger. More importantly, implementations for `MPI_Send` typically switch from buffered to synchronous mode above a certain message size. The former involves an extra data copy, while the latter avoids it because copying large amounts of data has high overhead. This "protocol switching" feature is seen in both OpenMPI and MPICH2. Due to such effects, instead of being a linear function of message size as in Eq. (4), communication time is rather *piece-wise linear*. Furthermore, depending on the mechanism, communications may be overlapped or not by computations or other communications, which ideally the simulation model should capture. Such synchronization and overlapping aspects can be partially accounted for by the classical LogP family of models [57, 58, 59, 60], and in particular LogGPS [60].

Figure 3 shows elapsed time vs. message size, using logarithmic scales for both axes, obtained from an experiment conducted with OpenMPI 1.6 on the *graphene* cluster of the Grid'5000 experimental testbed. This cluster

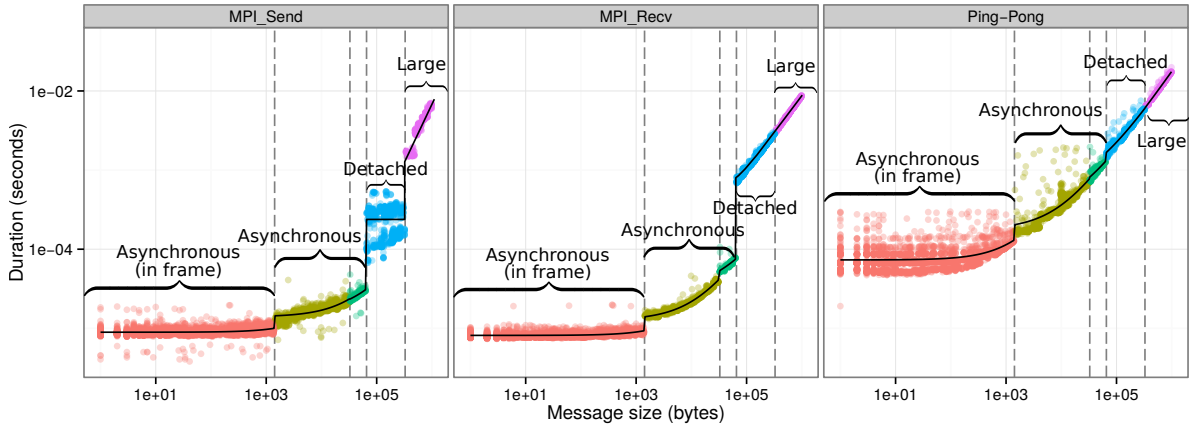


Figure 3: Duration of `MPI_Send`, `MPI_Recv`, and a ping-pong (a send immediately followed by a receive of the same size) vs. message size. Different modes can be seen depending on message size. Solid lines represent piece-wise linear regressions.

comprises 144 2.53 GHz Intel Xeon Quad-core X3440 nodes spread across four cabinets, and interconnected by a hierarchy of 10 Gigabit Ethernet switches. A full description of the interconnection network is available on-line[1]. The measurements were obtained as follows. To avoid measurement bias the message size is exponentially and randomly sampled between 1 byte and 100 MiB, for two types of experiments: "ping" and "ping-pong". The ping experiments aim at measuring the time spent in `MPI_Send` (resp. `MPI_Recv`) by ensuring that the receiver (resp. sender) is always ready to communicate. The ping-pong experiment consists in sending a message and immediately receiving a message of the same size, which allows us to measure the transmission delay. The goal is to study the behavior of MPI from the application's point of view, without any a priori assumptions about the way in which the MPI implementation switches between communication protocols depending on message size, which is in general difficult to determine based solely on the MPI configuration parameters.

Protocol switching effects are clearly seen in Figure 3. For messages below 32 KiB fully asynchronous communication is used, for messages above 256 KiB fully synchronous communication is used, and in between partially synchronous or "detached" communication is used. Each protocol leads to elapsed times that can be accurately modeled through linear regression. Up to three linear regressions, however, should be used for asynchronous communication depending on message size. For instance, a separate mode is required to capture accurately the case when the message is small enough to fit inside a single TCP frame. Overall, we have five distinct modes (modes 2 and 3 are almost identical for the `MPI_Send()` and the Ping-Pong results), for an overall behavior that is discontinuous and piece-wise linear. The simple linear model in the previous section would be reasonably accurate for small and large messages, but largely inaccurate in between (for more than 30% average error overall, with a worst case at 127%). Likewise, the classical LogP models [57, 58, 59, 60] do not model the piece-wise linear behavior accurately. The closest contender would be LogGPS [60], but it distinguishes between only two kinds of message sizes (small and large).

In [61] we have proposed a (non-necessarily continuous) piece-wise linear simulation model that can be instantiated for an arbitrary number of linear segments. This model was extended later in [62] to include the overlapping and synchronization aspects, as modeled by LogGPS. This new piece-wise linear model makes it possible to simulate accurately applications that use a wide range of message sizes [62]. The accuracy improvements due to the piece-wise linear model rather than the linear model are large enough to justify the increased number of parameters (2 parameters per mode). In addition, the value of these parameters are easily determined via straightforward experiments[2]. Furthermore, results in [61] show that the instantiation of the piece-wise linear model is robust: The instantiation computed on one cluster can be reused accurately for modeling other clusters with similar interconnect technology but different compute nodes. While these results are for two machines connected to the same switch, other results also show that our approach remains reasonably accurate when applied to a sequence of switches. This is important because large compute clusters are often organized as networks of switches.

An important implication of our accurate piece-wise linear model of point-to-point communication is that, when combined with the bandwidth sharing model described in the previous section, it leads to an immediate simulation model for collective communication operations. The collective operations are accessible to the SimGrid

[1] https://www.grid5000.fr/mediawiki/index.php/Nancy:Network

[2] The R code which allows to extract model parameters from a set of such measurements is available at http://mescal.imag.fr/membres/arnaud.legrand/research/smpi/smpi_loggps.php

user via the SMPI API (Figure 1). Just like any MPI implementation, the SMPI runtime implements collective communications as sets of point-to-point communications that may contend with each other on the network. This is to be contrasted with monolithic modeling of collective communications, as done in [1] for instance. These monolithic models rely on coarse approximations to model contention and/or on extensive calibration experiments that must be performed for each type of collective operation. Instead SimGrid provides implementations of collective communications based on current versions of both OpenMPI and MPICH. The current version of the SMPI runtime (v3.10) thus makes it possible to simulate the execution of MPI applications while accounting for network topology and contention in high-speed TCP networks. Extensions to other kind of topologies and other network technologies are underway. Results in [62] show that SimGrid can simulate collective communications effectively and has consistently a better predictive power than classical LogP-based models for a wide range of scenarios including both established HPC benchmarks and real applications. Section 6 presents a case study of the simulation of a full-fledged HPC application that performs many collective operations.

4.3. Extending the model for large networks

In a view to increasing the versatility of SimGrid, a worthwhile goal is to be applicable to peer-to-peer and volunteer computing domains in which hundreds of thousands of hosts on large wide-area networks must be simulated. Several simulators in that domain opt for packet-level simulation, e.g., OverSim [14], which can model contention in the network (close to the peers) as well as the network distance between the peers (which some peer-to-peer applications exploit). Our network model is applicable to these large networks, within the limits identified in Section 4.1, and is more scalable than packet-level simulation because more coarse-grain.

Regardless of whether one uses packet-level simulation or our model, one challenge is the instantiation of the simulation. It is not always possible for a user to instantiate a large network topology description in which every link is assigned a realistic latency and bandwidth value. It turns out that for many peer-to-peer and volunteer computing simulations there is no need to instantiate a complete network topology. Popular peer-to-peer simulators, such as PeerSim [13], implement instead a fixed-delay model by which a communication between two peers takes a fixed amount of time, or latency, which is deemed sufficient in terms of accuracy and has scalability advantages as well. A popular refinement of this model is to account for peer proximity by embedding the peers into a multi-dimensional space and using the distance between two points in this space as an estimate of the latency between two peers. Solutions have been proposed to realize such embedding in practice, such as the well-known decentralized Vivaldi [63] system. Our network model, as described in Section 4.1 can be used to simulate end-to-end latencies based on coordinates generated by systems like Vivaldi.

There is a large gap between the packet-level approach and the fixed-delay, coordinate-based approach. In particular, the latter does not account for network contention even at the peers themselves because it does not model bandwidths. SimGrid attempts to bridge that gap with a model that accounts for both end-to-end latencies and bandwidths, while still abstracting away the details of network topology so that the model is easily instantiable.

Previous research has shown that bandwidth at the edges of the network reflects the bandwidth available on full end-to-end paths. In other words, bandwidth bottlenecks are located within only a few hops of Internet end-points. For instance, Hu et al. [64] show that 60% of wide-area end-to-end paths hit a bandwidth bottleneck in the first or second hop. Similar findings have been reported for broadband access networks [65]. In [66] it is found that most end-to-end paths are limited by bandwidth at the end-points on the PlanetLab testbed [67]. These observations suggest a network model, which we term the "last mile" model, in which each host x is described by two bandwidths: an upload bandwidth β_x^{out} and a download bandwidth β_x^{in}. A communication from a host x to a host y is then allocated bandwidth $\beta_{xy} = \min(\beta_x^{out}, \beta_y^{in})$. Note that this model does not capture the fact that two end-points may be on the same local network, in which case the bandwidth available between these two end-points would be orders of magnitude larger than β_{xy} as computed above.

The network model in Section 4.1 can be trivially extended to support the last mile model. Furthermore, previous work shows that this model can be instantiated in practice. In [66], a decentralized algorithm is proposed that instantiates the model based on end-to-end bandwidth measurements on a real-world platform. Using a 308-host PlanetLab dataset for which full end-to-end bandwidth measurements are available, the model achieves good accuracy using a small number of measurements (each host only performs bandwidth measurements with 16 other hosts). In the end, the model is simple, instantiable, and more accurate than the fixed delay model. Furthermore, it is unified with the network model in SimGrid, thus further increasing versatility.

5. Versatile yet scalable simulations

Striving to make SimGrid more versatile (so that it can be used for, e.g., exascale HPC simulations as well as peer-to-peer simulations) has led us to tackling the scalability challenge along several directions. Scalability

is preconditioned on the use of a analytical simulation model, such as that described in Section 4. But three major scalability concerns, both in terms of memory footprint and CPU time, remain: (i) the efficiency of the implementation of the simulation model; (ii) the description of large platforms; and (iii) the simulation of large numbers of concurrent processes. In the next three sections we describe how SimGrid addresses these three concerns. We also provide quantitative comparisons to state-of-the-art domain-specific simulators for relevant case studies in the areas of volunteer computing, grid computing, and peer-to-peer computing. Section 6 demonstrates the scalability and accuracy of SimGrid simulations via a full-fledged case study in the HPC domain. Together, these case studies illustrate how our scalability solutions developed for specific domains can in fact be combined and applied to different domains. In other words, striving to make the simulator more versatile leads to scalability improvements across the board.

5.1. Efficient Simulation Kernel

As explained in Section 4, the base simulation model in SimGrid relies on a steady-state assumption to compute resource shares allocated to pending simulated activities. As a result, each time the set of these activities changes (a new activity is started, a current activity completes), the resource shares must be reevaluated, which amounts to solving a linear system of equations (Eq. (1)). The computed resource shares are then used to determine (i) by how much the simulated clock should be advanced and (ii) the progress of each pending activity.

Since simulated activities dynamically appear and disappear during the simulation, our implementation uses an ad hoc dynamic and sparse data structure to represent Eq. (1). The Max-Min fairness algorithm is straightforward, but it iterates several times over a dynamic set of variables. We first developed an implementation that was optimal in terms of number of operations, but it suffered from poor L1 and L2 cache reuse. To obtain an efficient cache-oblivious implementation we split the data structure in half so as to group together the few fields that are heavily used by the bandwidth sharing algorithm in contiguous arrays. The use of arrays instead of linked lists improves locality and hence the prefetch efficiency. This trimming of the data structures leads to improved cache utilization, and thus lowers simulation time, at the cost of significantly increased implementation complexity. Beside this data structure optimization, we have also implemented two algorithmic optimizations motivated by actual large-scale simulation scenarios [68], as detailed in the next two paragraphs.

Lazy activity updates. Originally, SimGrid was intended for the simulation of applications that comprise many communicating tasks running on computers connected by hierarchical networks. In this setting any event related to a simulated activity or resource can impact a large fraction of the other simulated activities and resources. However, when simulating large-scale platforms, such as those used for peer-to-peer or volunteer computing applications, most activities are independent of each other. In this case, reevaluating the full model becomes a performance bottleneck because all activities are examined even though many can simply be ignored most of the time. Our approach is thus to avoid solving the whole linear system in Eq. (1) by only recomputing the parts of it that are likely to be impacted by newly arrived or newly terminated activities. Furthermore, if between two resolution of the linear system only a few variables have changed, then only the state of the corresponding activities needs to be updated. Using a heap as a future event set, and efficiently detecting the set of variables that are impacted by activity removal and addition, we are able to lower the computational complexity of the simulation significantly. We term this technique "lazy updates," since the state of a simulated activity is modified only when needed.

Trace integration. Our second efficiency improvement targets the management of resources whose capacities change frequently. In SimGrid, the user can specify the capacity of a resource as a time-stamped trace to simulate fluctuating availability due to some out-of-band load (a capacity of zero means a downtime). The linear system in Eq. (1) must be reevaluated each time the capacity of a resource changes. In extreme scenarios, many such reevaluations may occur before a single activity completes, which would slow the simulation down unnecessarily. For instance, let us consider a situation in which the capacity of a resource is specified to change 100 times according to a user-specified time-stamped trace. Furthermore, let us assume that all pending activities still have large amounts of remaining work so that the earliest activity completion occurs after the 100th resource capacity change. In this case, 100 reevaluations of the linear system would take place even though would be possible to perform a single reevaluation. More formally, given current remaining work amounts, one can compute the next activity completion date given all future resource capacity values before this date. This computation can be performed efficiently using "trace integration." Essentially, instead of storing a trace as capacity values, one stores its integral. Finding the last resource capacity change before the next activity completion can then be performed using a binary search, i.e., with logarithmic time complexity.

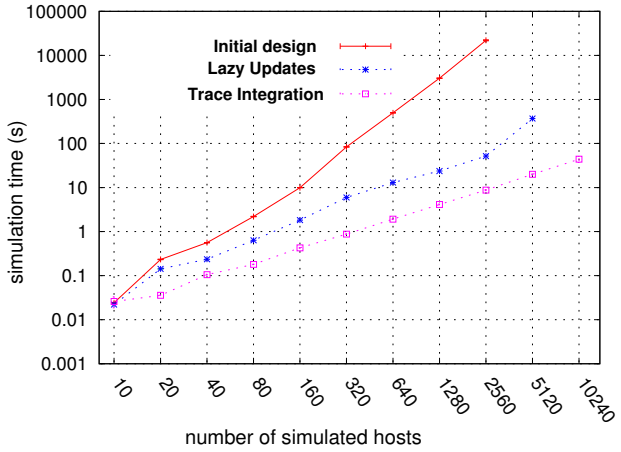


Figure 4: Simulation time vs. number of simulated hosts for a volunteer computing simulation using the initial design, with lazy updates, and with lazy updates and trace integration.

Case study: volunteer computing simulation

The scalability enhancements described in this section were initially motivated by the need to simulate large volunteer computing systems efficiently [68]. Let us consider a volunteer computing scenario with N hosts. Each host computes sequentially P tasks, and the compute rate of each host changes T times before the completion of the simulation. With our original implementation the time complexity of this simulation is $O(N^2(P+T))$. With lazy activity updates it becomes $O(N(P+T)\log N)$, and $O(NP(\log(N)+\log(T)))$ when adding trace integration. We have implemented such a simulation, using traces of MFlop/sec rates for SETI@home hosts available from [69]. Compute tasks have uniformly distributed random compute costs in MFlop between 0 and 8.10^{12} (i.e., up to roughly one day for a median host). Note that such simulation scenarios are commonplace when studying volunteer computing, and in fact this particular scenario was suggested to us by the authors of [70] to highlight scalability issues in previous versions of SimGrid. Figure 4 shows simulation time measured on a 2.2 GHz AMD Opteron processor vs. N for the initial design, the addition of lazy activity updates, and the addition of trace integration, using a logarithmic scale on the vertical axis. Results make it plain that both proposed improvements decrease simulation time dramatically. For instance, for a simulation with 2,560 hosts, the simulation time is almost at 3h without the enhancements, around 1min with lazy updates, and under 10 s with lazy updates and trace integration. A comparison with the state-of-the-art SimBA simulator [10], based on timing results published therein and the use of a similar benchmark machine, shows that with our improvements SimGrid achieves simulation times more than 25 times faster. We refer the interested reader to [68] for more details on the experimental setup. This is an important result given that SimGrid is more versatile than SimBA. In fact, the behaviors of the network and the software are simulated in much more details in SimGrid (i.e., flow-level model of TCP, programmatic specification) than in SimBA (i.e., fixed latency, finite automata).

The trace integration mechanism is specific to CPU simulation, but the lazy update mechanism applies across all resources and activities. To quantify the impact of lazy updates on the speed of network simulation, Figure 5 shows simulation times when simulating various numbers of flows (10, 50, 100, or 150) that are opened and closed at random dates between random pairs of nodes for 1,000 seconds of simulated time, for three representative platforms. A randomized factorial set of experiments with 50 measurements for each combination is run on a 3.3 GHz Core i7 processor and we report the 95% confidence intervals of the average time needed to perform the simulation (platform description parsing time is not included). The first platform consists of 1,740 independent hosts each with its own upstream link and downstream link, using the "last mile" model discussed in Section 4.3. When peer A communicates with peer B, a network flow using the upstream link of A and the downstream link of B is created and the latency of this flow is computed from the link latencies and the Vivaldi [63] network coordinates of the peers. This platform is thus very loosely coupled, and as such we see in the leftmost graph in Figure 5 that the use of lazy updates reduces the simulation time significantly. The second platform comprises 90 hosts and 20 routers and was created with the Tiers algorithm [71], which uses a three-step space-based hierarchical approach. The resulting topology is hierarchical with low bisection bandwidth, and has thus both global (in the core of the network) and local (on the edges of the network) bottleneck links. The third platform comprises 200 nodes and is generated with the Waxman model [72] and the BRITE generator [73]. In this platform there are more alternate network paths but the unstructured communication patterns of the simulated application leads here also to interference among flows in the network. The results in Figure 5 for the second and third platforms show that

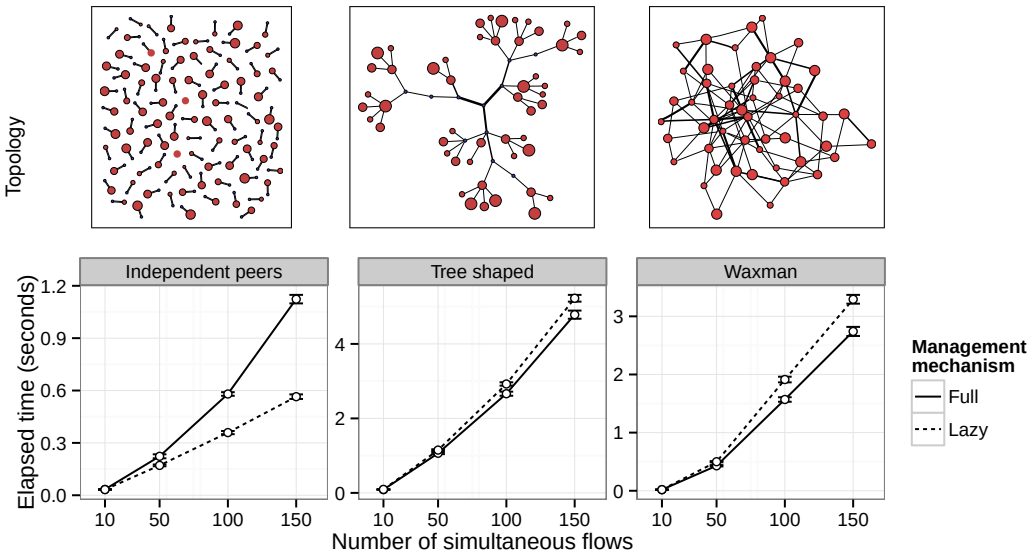


Figure 5: Simulation time vs. number of network flows for three different network topologies without and with lazy updates. On loosely connected platforms (e.g., independent peers), the lazy updates mechanism reduces simulation time significantly. On more tightly coupled platforms it can increase simulation time.

lazy updates actually increase simulation time. This is because in these platforms the probability that a random flow interferes with another is high. As a result, our lazy updates implementation suffers from some overhead when determining that the resource shares of most flows needs to be recomputed. For this reason, lazy updates can be deactivated by the user. However, since lazy updates only incur marginal slowdowns but bring significant speedup when there is locality in the communication patterns, SimGrid enables them by default.

5.2. Scalable platform descriptions

As stated in Section 2.3, the most expressive way to describe a network interconnect is to describe the routing table of each simulated network element explicitly. Unfortunately, this method is memory consuming. The last-mile model in Section 4.3 provides a partial solution that is applicable in some domains. More generally, SimGrid uses a scalable, efficient, and modular network representation technique, which also drastically reduces the platform description burden placed on users.

SimGrid's platform representation exploits the *hierarchical* structure of real-world (large-scale) network infrastructures [74], relying on the concept of autonomous systems (AS), including local networks as well as the classical Internet definition. In addition, the representation is recursive within each AS so that regular patterns can be exploited whenever possible. SimGrid provides stock implementations of well-known routing schemes, including Dijkstra, Dijkstra with cache, Floyd, Flat (i.e., full routing table), Empty routing with Vivaldi network coordinates (see Section 4.3), and cluster (i.e., a regular topology where each node has its own private links and communicates with the others through an additional shared link). For the time being, and to favor scalability, SimGrid assumes that the routing is static over time. This assumption is reasonable (see for instance the study in [75], which shows that less than 20% of Internet paths change in a 24-hour period). Besides, routing changes in real-world networks are known to affect traffic on backbone links. Usually, these links are not communication bottlenecks. Therefore, routing changes can likely be ignored without a large impact on simulation accuracy. Figure 6 shows an example hierarchical network representation in SimGrid.

Each AS declares a number of gateways, which are used to compute routes between ASes comprised within a higher-level AS. This mechanism is used to determine routes between hosts that belong to different ASes: simply search for the first common ancestor in the hierarchy and resolve the path recursively. The network representation and this route computation method provide a compact and effective representation for hierarchical networks. Since real-world networks are not purely hierarchical, SimGrid provides "bypassing" rules that can be used to declare alternate routes between ASes manually.

The above semantic principles of network representation are implemented by the user via an XML file. For convenience, SimGrid provides a set of XML tags that simplify the definition of two standard and ubiquitous ASes: homogeneous clusters and sets of Internet peers. The cluster tag creates a set of homogeneous hosts interconnected through private links and a backbone, which all share a common gateway. The peer tag allows for the easy creation of peer-to-peer platforms by defining at the same time a host and a connection to the rest of the network (with

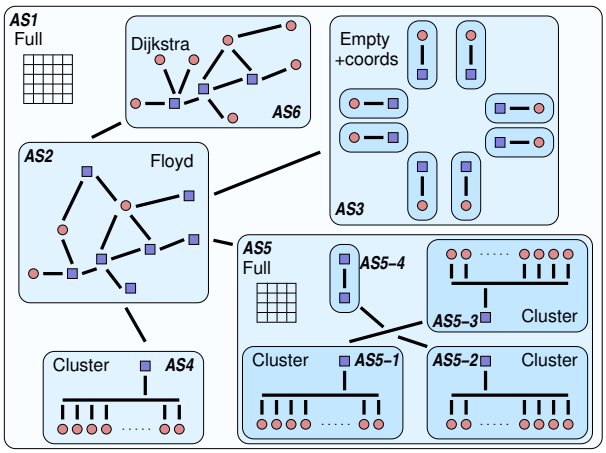


Figure 6: Example hierarchical network representation of an AS, AS1. Circles represent compute resources and squares represent network routers. Bold lines represent communication links. Routing schemes are indicated for each AS. AS2 models the core of a national network interconnecting a small cluster (AS4), a larger hierarchical cluster (AS5), a subset of a LAN (AS6), and a set of peers scattered across the Internet (AS3).

different upload and download characteristics and network coordinates, as explained in Section 4.3). SimGrid also provides an API for generating in-memory network descriptions directly without requiring an XML file.

Case study: grid computing simulations

SimGrid's network description approach makes it possible to describe large platforms with low memory footprint and without significant computational overhead. For instance, it allows us to represent the full Grid'5000 platform [76] (10 sites, 40 clusters, 1,500 nodes) with only 61 KiB. By comparison, the flat representation with SimGrid v3.2 required 1,065 MiB [77]. It takes more than 4 minutes to parse the flat representation on a 1.6 GHz Intel Core2 Duo with 5 GiB of memory and an SSD drive while the current representation is parsed in less than 150 milliseconds.

We now compare the scalability of SimGrid to the widely used GridSim toolkit [6] (version 5.2 released on Nov. 25, 2010). The experimental scenario is a simple master-worker execution where the master distributes T fixed size tasks to W workers in a round-robin fashion. In GridSim we did not define any network topology, hence only the output and input baud rates are used to determine data transfer rates. By contrast, with SimGrid we used the aforementioned Grid'5000 network representation, which models clusters and their cabinets as well as the wide area network interconnecting the different sites. Furthermore, SimGrid uses the flow network model described in Section 4. Experiments were conducted using a 2.4 GHz Intel Xeon Quad-core with 8 GiB of RAM. We refer the interested reader to [74] for more details on the experimental setup.

	Simulation Time	Peak Memory Footprint
GridSim	$56\,\text{ms} \times W + 14\,\text{ns} \times T^2$	$2.5\,\text{GiB} + 226\,\text{KiB} \times W + 3\,\text{KiB} \times T$
SimGrid	$0.1\,\text{ms} \times W + 26\,\mu\text{s} \times T$	$5.2\,\text{MiB} + 80\,\text{KiB} \times W$

Table 2: Polynomial fits of simulation times and peak memory footprints of GridSim and SimGrid for a master-worker simulation with W workers and N tasks. The simulation time is quadratic with T in GridSim while it is linear in SimGrid. The peek memory footprint of GridSim is several orders of magnitude larger than that of SimGrid.

The number of tasks, T, is uniformly sampled in the $[1; 500{,}000]$ interval and the number of workers, W, is uniformly sampled in the $[100; 2{,}000]$ interval We perform 139 experiments for GridSim and 1,000 for SimGrid (as it was significantly faster), and measure the wall-clock time (in seconds) and the memory consumption (using the Maximum Resident Set Size in KiB as a measurement). As expected, the size (input and output data, and amount of computation) of the tasks have no influence. Experimental results are shown as polynomial fits in Table 2. The goodness of fit is high (all coefficients of determinations, or R^2, for all fits are above 0.9972).

The simulation time for GridSim is quadratic in T and linear in W. Surprisingly, GridSim's memory footprint is not polynomial in T and W. Rather, it appears to be piece-wise linear in both (with a very steep slope at first, and less steep as values increase). Furthermore there are a few outlier points that exhibit particularly low or high memory usages (leading to $R^2 = 0.9871$). This is likely explained by the Java garbage collection. For this reason, in Table 2 we only report results for scenarios where T is larger than 200,000, which removes most outliers.

Analyzing the results for SimGrid shows that its simulation time and memory footprint are stable and several orders of magnitude lower. The results reported in Table 2 mean that 5.2 MiB are required to represent the Grid'5000 platform and the internals of SimGrid, with a payload of 80 KiB per worker. By comparison GridSim uses 2.5 GiB with an additional 300 MiB payload per worker. Furthermore, in SimGrid T has no impact on the memory footprint, which is not the case in GridSim. We conclude that SimGrid, with its flow network model and a fine-detailed network topology, is several orders of magnitude faster and more memory efficient than GridSim, with its delay-based model and no network topology. For instance, while GridSim requires more than one hour and 4.4 GiB of memory to simulate the execution of 500,000 tasks with 2,000 workers, SimGrid performs this same simulation in less than 14 seconds and with only 165 MiB.

5.3. Efficient simulation of concurrent processes

SimGrid allows users to describe the simulated application programmatically as a set of independent but communicating concurrent processes. The goal is to allow users to implement the simulated application in a way that is similar to but simpler than the way in which a real application would be implemented. Due to the optimizations described in Section 5.1, for many large-scale simulations the most computationally intensive portion of the simulation is not the evaluation of the simulation model, but instead the execution and the synchronization of the simulated processes! As a result, increasing scalability requires going beyond vanilla implementations, e.g., based on threads and standard synchronization primitives.

Since the simulation models in SimGrid can be computed quickly, it is possible and in fact efficient to have a unique execution context (such as a thread) handle all the simulation model computations. We call this context the *core context*, and it interacts with the execution contexts of the simulated concurrent processes. This has led to the layered design shown in Figure 1. At the bottom is the SURF component that runs in the core context and deals with the simulation of the resources and of their usage by the activities issued by the simulated concurrent processes. At the top are the concurrent processes themselves, implemented as user code that places calls to a SimGrid API (MSG or SMPI) to define activities. In between is a *synchronization kernel*, SIMIX, that mediates every interaction between the simulated processes and the core context.

The synchronization kernel is conceptually close to the kernel of a classical operating system and it emulates a system call interface called *simcalls*. Simcalls are used by simulated processes to interact with the core context. When a simulated process issues a simcall the request and its arguments are stored in a private memory location. The process is then blocked until the completion of the request (e.g., completion of the corresponding simulated activity). When all user processes are blocked in this manner control is passed to the core context. The core context handles the requests sequentially in an arbitrary but deterministic order based on process IDs, and it is the only context that accesses the simulation state. A sequential core context makes for simplified simulation logic due to vastly reduced numbers of context switches between the core context and the simulated processes. To the best of our knowledge it is the first time that this classical OS design is applied to distributed system simulation. An alternate design in which simulated entities actively interact with each other, such as that used for instance in GridSim [6], may seem more intuitive but leads to more complex simulation logic due to multi-step interactions between processes/threads.

Our design is scalable only if mechanisms are available to execute thousands or even millions of processes on a single host (standard virtual machine techniques cannot be used to execute our simulated processes as at most dozens of virtual machines instances can run efficiently on a host). The use of regular threads seems like a natural approach, with the code of each simulated concurrent process running in its own thread. But with standard threads, one can scale up to "only" a few thousands simulated processes, thus severely limiting the scale of the simulation. For instance, GridSim, which uses threads, cannot simulate more than 10,000 processes/hosts [78]). Instead, we employ cooperative, light-weight, non-preemptive threads (known as *continuations*). They are ideally suited to our needs since our synchronization kernel has to finely control the scheduling of the simulated processes anyway. Additionally, they are much simpler to implement than regular threads. The Windows operating system provides such light-weight execution contexts as *fibers*, while they are called *ucontexts* (for user-contexts) on Unix operating systems, including Mac OSX. In SimGrid we have aggressively re-implemented a similar mechanism directly in assembly so as to remove a costly and unnecessary system call found in standard implementations.

Case study: scalable peer-to-peer simulation

In [79], we compare the scalability of SimGrid for a peer-to-peer simulated scenario to that of two popular and reported-to-be-scalable simulators in that domain: OverSim [14] and PeerSim [13]. Figure 7 shows the simulation time of the Chord protocol [80] vs. the number of simulated peers. For SimGrid and OverSim we use the experimental scenario initially proposed in [14]: each peer joins the Chord ring at time $t = 0$, then performs a stabilize operation every 20 seconds, a fix fingers operation every 120 seconds, and an arbitrary lookup request

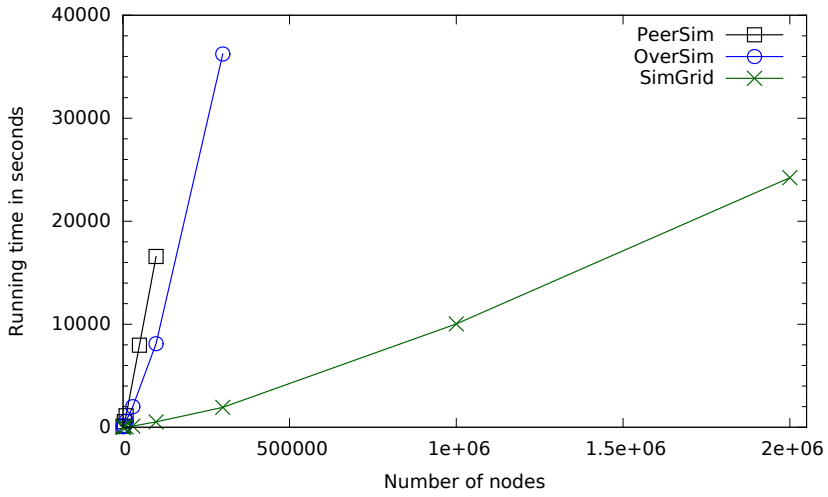


Figure 7: Simulation time vs. number of peers for a Chord simulation with SimGrid (with constant and precise network models), OverSim (with a simple underlay and using the OMNeT++ bindings), and PeerSim.

every 10 seconds. The simulation ends at $t = 1000$ seconds. For PeerSim, the implementation that is publicly available does not make this experimental scenario possible since there are not stabilize or fix fingers operations. So in the PeerSim experiments a single lookup is generated every 120 seconds.

We use the Chord implementations that are publicly available for OverSim and PeerSim, while we have implemented the Chord protocol ourselves for SimGrid. Therefore, there may be differences (parameters, features, optimizations, or even bugs) among the three implementations of the protocol. To ensure that experiments are comparable in spite of such differences, we tune the simulated scenario parameters to make sure that the numbers of application messages exchanged during the simulation, and thus the load on the simulator, are comparable across experiments (10,000 peers, 25 million messages). More specifically, we conservatively ensure that more messages are exchanged in the SimGrid simulation than in the OverSim and PeerSim simulations. Note that the three simulators in our comparisons record different information (i.e., simulation event traces), leading to different tracing overheads. However, as seen hereafter, our results show orders of magnitude improvements for SimGrid over its competitors.

Experiments were conducted on one core of a two-CPU 1.7 GHz AMD Opteron 6164 HE (12 cores per CPU) with 48 GiB of RAM running Linux. OverSim (v20101103) is implemented in C++ (gcc v4.4.5), SimGrid (SimGrid v3.7-beta, git revision 918d6192) in C (gcc v4.4.5), and PeerSim (v1.0.5) in Java (HotSpot JVM v1.6.0-26). In our experiments, we configured PeerSim so that its simulation model assumes that every communication takes a uniform random amount of time. We configured OverSim to use a simple model in which communication times are based on the Euclidean distance between processes (instead of the less scalable OMNeT++ bindings). By contrast, for this experiment SimGrid uses its flow-level model that accounts for more complex network behavior (i.e., contention and TCP congestion avoidance). Therefore, the simulation model of SimGrid subsumes and is thus strictly more realistic than the simulation models in OverSim and PeerSim. We refer the interested reader to [79] for full details.

Results show that the largest scenario that we managed to run in less than 12 hours using PeerSim was for 100,000 peers (4h36min). This poor result is likely due to the Java implementation. With OverSim, we managed to simulate 300,000 peers in 10 hours. With SimGrid we were able to simulate 2,000,000 peers in 6h43min. Simulating 300,000 peers took 32 minutes. The memory footprint for simulating 2 million peers with SimGrid was about 36 GiB, which amounts to 18 KiB per peer, including 16 KiB for the stack devoted to the user code.

We conclude that SimGrid leads to drastic scalability improvements when compared to state-of-the-art peer-to-peer simulators, even though these simulators were designed specifically for scalable simulations. For instance, SimGrid is 15 times faster than OverSim and can simulate scenarios that are 10 times larger even though it uses much more sophisticated (network) simulation models. The reasons for these large performance improvements over domain-specific simulators are the various optimizations/designs of the simulation engine described in this work, which were driven by the need for simulation versatility.

6. Accurate and scalable HPC simulations

In this section we present a case study in the HPC domain. Developing a simulator that makes it possible to simulate a few standard HPC benchmarks with reasonable accuracy requires a fair amount of effort, and has merit to demonstrate the potential of the simulator. However, the ultimate goal is for the simulator to be usable (i.e., accurate and scalable) for simulating real applications. For this reason, we have evaluated SMPI, the component of SimGrid that makes it possible to simulate MPI applications, for both benchmarks and complex applications, including the full LinPACK suite [81], the Sweep3D [82] benchmark, the BigDFT Density Functional Theory application [83], and the SpecFEM3D geodynamics application [84] that is part of the PRACE benchmark. SMPI is tested on a daily basis for 80% of the MPICH2 test suite and against a large subset of the MPICH3 test suite.

In [62] we have demonstrated the ability of SMPI to simulate a real, large, and complex MPI application and we report here a part of these results to illustrate the effectiveness of the models presented in Section 4.2. To this end, we use BigDFT, which is the sole electronic structure code based on systematic basis sets that can use hybrid supercomputers and has good scaling (95% efficiency with 4,096 nodes on the Curie supercomputer). For this reason, BigDFT was selected as one of the eleven scientific applications in the Mont-Blanc project [85]. The goal of this project is to assess the potential of low-power embedded components, such as commercially available ARM processing and network components, for building exascale clusters. The first Mont-Blanc prototype is expected to become available during 2014. It will include Samsung Exynos 5 Dual Cortex A15 processors with an embedded Mali T604 GPU and will be using Ethernet for communication. To evaluate the applications before the prototype is available, a small cluster of ARM systems-on-chip was built at the Barcelona Supercomputing Center, Tibidabo [86], which uses NVIDIA Tegra2 chips, each with a dual-core ARM Cortex A-9 processor. The PCI Express support of Tegra2 is used to connect a 1 GbE NIC, and the boards are interconnected hierarchically using 48-port 1 GbE switches. The application execution results that are presented in this section have been obtained on Tibidabo. The OpenMP and GPU extensions of BigDFT were disabled so as to focus on the behavior of the MPI operations. We used MPICH 3.0.4 [87] and we refer the interested reader to [62] for more details on the experimental setup.

BigDFT alternates between computation bursts and intensive collective communications. The collective operations that are used are diverse and can change depending on the instance, hence requiring accurate modeling of a broad range of collective communications for the purpose of simulating BigDFT executions. BigDFT can be simulated with SMPI with minimal source code modification. BigDFT has a large memory footprint, which precludes running it on a single machine. However, thanks to the memory folding and partial execution techniques implemented as part of SMPI (see [61]), we were able to simulate the execution of BigDFT with 128 processes, with a peak memory footprint estimated at 71 GiB, on a 1.6 GHz Intel Core2 Duo processor with less than 2.5 GiB of RAM.

Figure 8 shows parallel speedup vs. number of compute nodes, as measured on the Tibidabo cluster for an instance of the BigDFT application. This instance has a relatively low communication to computation ratio in spite of Tibidabo's relatively slow compute nodes (around 20% of time is spent communicating when using 128 nodes), and uses the following collective operations: `MPI_Alltoall`, `MPI_Alltoallv`, `MPI_Allgather`, `MPI_Allgatherv` and `MPI_Allreduce`. This particular instance is a difficult case for simulation. This is because the large number of collective communication operations severely limits the scalability of the application, thus requiring precise simulation of these operations. Yet, accurately assessing such scalability limits in simulation is crucial for deciding how to provision a platform before it is actually purchased and deployed.

In addition to the real speedup measurements, Figure 8 shows the speedup computed based on simulation results obtained with SimGrid, as well as the speedup computed according to the LogGPS model [60]. As expected, both are more optimistic than the real execution. However, while SimGrid tracks the trend of the real measurements well (within 8%), LogGPS is overly optimistic (up to 40% error). As explained in Section 4.2, unlike models from the LogP family, SimGrid relies on a model that combines flow-level models (to account for contention on arbitrary network topologies), a piece-wise linear model (to model the protocol switching feature of MPI implementations), and a LogP model (to model the computation/communication overlap and the communication synchronization semantic). The results in Figure 8 show that this model is significantly more accurate than the LogGPS model. In particular, unlike LogGPS, it successfully accounts for the slowdown of BigDFT incurred by the hierarchical and irregular network topology of the Tibidabo platform.

One interesting question is whether the higher accuracy of SimGrid when compared to the use of the LogGPS model comes at an acceptable loss in simulation scalability. In other words, how long does the SimGrid simulation take? To answer this question we compare the scalability of SimGrid to that of LogGOPSim 1.1 [2], a recent simulator designed specifically to simulate the execution of MPI applications on large-scale HPC systems. LogGOPSim relies on the LogGPS model. We use the same experimental setting described in Section 4.1.2 of [2],

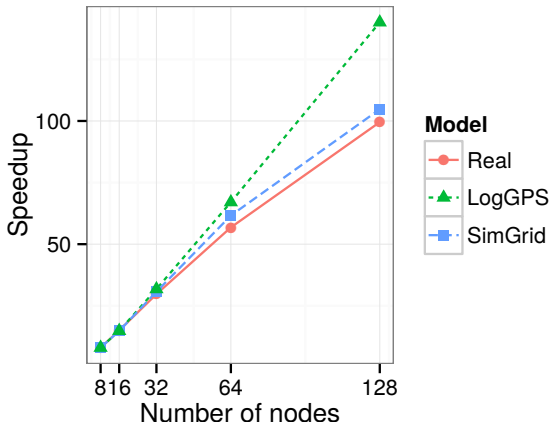


Figure 8: Parallel speedup vs. number of compute nodes for BigDFT on Tibidabo, for real executions, SimGrid simulations, and LogGPS models.

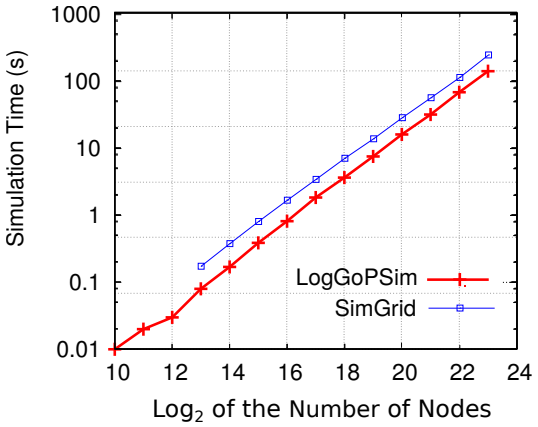


Figure 9: Simulation time vs. number of simulated nodes for SimGrid and LogGOPSim when simulating a binomial broadcast.

i.e., the execution of a binomial broadcast on various numbers of nodes. Unfortunately, as the input traces used therein were not available, we compare our results to the published results instead of reproducing the experiments with LogGOPSim. We use SimGrid v3.7-beta (git revision 918d6192 and gcc v4.4.5) to simulate a platform interconnected with a hierarchy of 64-port switches. The binomial broadcast is implemented with the SimDAG API. The original evaluation of LogGOPSim was done on a 1.15 GHz Opteron workstation with 13 GiB of memory. Instead, we use one core of a node with two AMD Opteron 6164 HE 12-core CPUs at 1.7 GHz with 48 GiB of memory, which we scale down to 1 GHz to allow for a fair comparison. We refer the interested reader to [74] for more details on the experimental setup.

Figure 9 shows simulation time vs. the number of simulated nodes for both SimGrid and LogGOPSim. While using significantly more elaborate platform and communication models, and thus leading in general to much improved accuracy (see Figure 8), SimGrid is only about 75% slower than LogGOPSim. This percentage slowdown is almost constant up to large scales with millions of simulated nodes. SimGrid's memory usage for 2^{23} nodes in this experiment is 15 GiB, which is larger than what is achieved in [2] (whose experiments were conducted on a machine with 13 GiB of RAM). The incurred scalability penalties in terms of simulation time and memory footprint are likely worthwhile for most users given the large improvement in simulation accuracy.

Our broad conclusion, from this and the other case studies presented in this work, is that a simulator can have both high accuracy and high scalability. It is interesting to note that SimGrid initially targeted grid computing applications, which led to the development of flow-level network models that account for network contention (Section 4.1). SimGrid then began being used for peer-to-peer and volunteer computing simulations, which required an optimization of the simulation model evaluation algorithm (Section 5.1), the design of efficient platform models (Section 4.3) and representations (Section 5.2), and the optimization of the simulation of concurrent processes (Section 5.3). Targeting the simulation of HPC systems required improving the network model (Section 4.2). In the end, while these advances were motivated by various domains, their benefits are felt across all these domains. The results achieved for the HPC case study presented in this section would not have been possible had not all these advances been accomplished within a single versatile simulation framework.

7. Conclusion

In this article we have given an overview of the SimGrid project and have highlighted recent scientific and engineering advances in the context of this project. These advances have led to improvements in both simulation accuracy and scalability. The important lesson is that most of these advances have been successful and yet motivated by the need to push the versatility of SimGrid beyond its original target domain. This is contrary to the popular wisdom that specialization allows for better simulations. And indeed, as seen in the results we have presented, SimGrid outperforms several more specialized state-of-the-art simulators. We claim that this is not in spite of its increased versatility but because of it. A clear benefit of this versatility is that it is now possible to conduct SimGrid simulations that cross multiple domains. For instance, one can combine peer-to-peer and high-performance computing simulations (e.g., to simulate a set of commodity clusters interconnected via high-performance backbones augmented with a large set of Internet-connected peers).

An important future direction for SimGrid is the development of new or improved simulation models for (i) memory hierarchies, which have a large impact on the performance of HPC applications; (ii) storage resources, which are often a performance bottleneck in HPC and cloud environments; and (iii) power consumption of the simulated application/platform, which is an overriding concern for all large-scale platforms be they cloud infrastructures or petascale/exascale HPC platforms. Another direction relates to the design and analysis of simulation experiments. In many fields, conducting experiments to acquire sample data is expensive (e.g., industrial processes). Given the relatively low number of samples, practitioners must rely on sound statistical techniques. By contrast, because computer simulation experiments are cheap, most computer scientists acquires large numbers of samples via thousands of simulation experiments with the informal rationale that statistical significance is achieved by large numbers. As a result, although a broad generalization is likely unfair, computer scientists often seem to use poor statistical techniques. Our own recent use of solid statistical techniques has, unsurprisingly, proved extremely beneficial both in terms of result confidence and of simulation times. Popularizing the use of these techniques, by providing a simulation design and analysis framework as part of SimGrid, would represent a major step toward better scientific practice in this field.

In experimental sciences the ability to reproduce published results is the necessary foundation for obtaining universal and enduring knowledge, and part of the "Open Science" approach widely adopted in fields such as physics or chemistry. To date, most simulation results in the parallel and distributed computing literature are obtained with simulators that are ad hoc, unavailable, undocumented, and/or no longer maintained. For instance, in 2013, the authors in [24] point out that out of 125 recent papers they surveyed that study peer-to-peer systems, 52% use simulation and mention a simulator, but 72% of them use a custom simulator. As a result, most published simulation results are impossible to reproduce by researchers other than the authors. There is thus a strong need for recognized simulation frameworks by which simulation results can be reproduced and further analyzed. Our goal is for SimGrid to fill this need, which is why SimGrid welcomes contributors and is publicly available at http://simgrid.org/.

Acknowledgements

The authors would like the thank SimGrid team members and collaborators whose work has contributed ideas and content for this article: Olivier Beaumont, Laurent Bobelin, Pierre-Nicolas Clauss, Augustin Degomme, Bruno Donassolo, Lionel Eyraud-Dubois, Stéphane Genaud, George Markomanolis, David González Márquez, Pierre Navarro, Christian Rosa, Lucas Schnorr, Mark Stillwell, Christophe Thiéry, Pedro Velho, Jean-Marc Vincent, Young Joon Won.

This work has been supported by *ANR* (French National Agency for Research) through project references ANR 08 SEGI 022 (USS SimGrid) and ANR 11 INFRA 13 (SONGS), by *CNRS* (French National Center for Scientific Research) through PICS 5473 grant, and by *Inria* through an *ADT* (software and technological development actions) and internship programs.

Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see https://www.grid5000.fr).

[1] M. Tikir, M. Laurenzano, L. Carrington, A. Snavely, PSINS: An Open Source Event Tracer and Execution Simulator for MPI Applications, in: Proc. of the 15th International Euro-Par Conference on Parallel Processing, no. 5704 in LNCS, Springer, 2009, pp. 135–148.

[2] T. Hoefler, T. Schneider, A. Lumsdaine, LogGOPSim - Simulating Large-Scale Applications in the LogGOPS Model, in: Proc. of the ACM Workshop on Large-Scale System and Application Performance, 2010, pp. 597–604.

[3] G. Zheng, G. Kakulapati, L. Kalé, BigSim: A Parallel Simulator for Performance Prediction of Extremely Large Parallel Machines, in: Proc. of the 18th International Parallel and Distributed Processing Symposium (IPDPS), 2004.

[4] R. Bagrodia, E. Deelman, T. Phan, Parallel Simulation of Large-Scale Parallel Applications, IJHPCA 15 (1) (2001) 3–12.

[5] W. H. Bell, D. G. Cameron, A. P. Millar, L. Capozza, K. Stockinger, F. Zini, OptorSim - A Grid Simulator for Studying Dynamic Data Replication Strategies, IJHPCA 17 (4) (2003) 403–416.

[6] R. Buyya, M. Murshed, GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing, Concurrency and Computation: Practice and Experience 14 (13-15) (2002) 1175–1220.

[7] S. Ostermann, R. Prodan, T. Fahringer, Dynamic Cloud Provisioning for Scientific Grid Workflows, in: Proc. of the 11th ACM/IEEE International Conference on Grid Computing (Grid), 2010, pp. 97–104.

[8] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, R. Buyya, CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms, Software: Practice and Experience 41 (1) (2011) 23–50.

[9] A. Núñez, J. Vázquez-Poletti, A. Caminero, J. Carretero, I. M. Llorente, Design of a New Cloud Computing Simulation Platform, in: Proc. of the 11th International Conference on Computational Science and its Applications, 2011, pp. 582–593.

[10] M. Taufer, A. Kerstens, T. Estrada, D. Flores, P. J. Teller, SimBA: A Discrete Event Simulator for Performance Prediction of Volunteer Computing Projects, in: Proc. of the 21st International Workshop on Principles of Advanced and Distributed Simulation, 2007, pp. 189–197.

[11] T. Estrada, M. Taufer, K. Reed, D. P. Anderson, EmBOINC: An Emulator for Performance Analysis of BOINC Projects, in: Proc. of the Workshop on Large-Scale and Volatile Desktop Grids (PCGrid), 2009.

[12] D. Kondo, SimBOINC: A Simulator for Desktop Grids and Volunteer Computing Systems, Available at http://simboinc.gforge.inria.fr/ (2007).
[13] A. Montresor, M. Jelasity, PeerSim: A Scalable P2P Simulator, in: Proc. of the 9th International Conference on Peer-to-Peer, 2009, pp. 99–100.
[14] I. Baumgart, B. Heep, S. Krause, OverSim: A Flexible Overlay Network Simulation Framework, in: Proc. of the 10th IEEE Global Internet Symposium, IEEE, 2007, pp. 79–84.
[15] H. Casanova, A. Legrand, M. Quinson, SimGrid: a Generic Framework for Large-Scale Distributed Experiments, in: Proc. of the 10th IEEE International Conference on Computer Modeling and Simulation (UKSim), 2008.
[16] W. Gropp, E. Lusk, A. Skjellum, Using MPI: Portable Parallel Programming with the Message Passing Interface, 2nd Edition, Scientific And Engineering Computation Series, MIT Press, 1999.
[17] J. Zhai, W. Chen, W. Zheng, PHANTOM: Predicting Performance of Parallel Applications on Large-Scale Parallel Machines Using a Single Node, in: Proc. of the 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, 2010, pp. 305–314.
[18] E. Totoni, A. Bhatele, E. Bohm, N. Jain, C. Mendes, R. Mokos, G. Zheng, L. Kalé, Simulation-based Performance Analysis and Tuning for a Two-level Directly Connected System, in: Proc. of the 17th International Conference on Parallel and Distributed Systems, 2011, pp. 340–347.
[19] P. Dickens, P. Heidelberger, D. Nicol, Parallelized Direct Execution Simulation of Message-Passing Parallel Programs, IEEE Transactions on Parallel and Distributed Systems 7 (10) (1996) 1090–1105.
[20] R. Riesen, A Hybrid MPI Simulator, in: Proc. of the IEEE International Conference on Cluster Computing, 2006.
[21] E. León, R. Riesen, A. Maccabe, P. Bridges, Instruction-Level Simulation of a Cluster at Scale, in: Proc. of the International Conference for High Performance Computing and Communications (SC), 2009.
[22] K. Ranganathan, I. Foster, Decoupling Computation and Data Scheduling in Distributed Data-Intensive Applications, in: Proc. of the 11th IEEE International Symposium on High Performance Distributed Computing (HPDC), 2002, pp. 352–358.
[23] Berkeley Open Infrastructure for Network Computing, http://boinc.berkeley.edu (2013).
[24] A. Basu, S. Fleming, J. Stanier, S. Naicken, I. Wakeman, V. K. Gurbani, The State of Peer-to-peer Network Simulators, ACM Computing Survey. 45 (4) (2013) 46.
[25] A. Varga, The OMNeT++ Discrete Event Simulation System, in: Proc. of the 15th European Simulation Multiconference (ESM), 2001.
[26] T. M. Gil, M. F. Kaashoek, J. Li, R. Morris, J. Stribling, P2PSim, a Simulator for Peer-to-Peer Protocols, Available at http://pdos.csail.mit.edu/p2psim/ (2005).
[27] P. García, C. Pairot, R. Mondéjar, J. Pujol Ahulló, H. Tejedor, R. Rallo, PlanetSim: A New Overlay Network Simulation Framework, in: Proc. of the 4th International Workshop on Software Engineering and Middleware (SEM), Vol. 3437 of LNCS, Springer, 2004, pp. 123–136.
[28] S. Sioutas, G. Papaloukopoulos, E. Sakkopoulos, K. Tsichlas, Y. Manolopoulos, A Novel Distributed P2P Simulator Architecture: D-P2P-Sim, in: Proc. of the 18th ACM Conference on Information and Knowledge Management (CIKM), 2009, pp. 2069–2070.
[29] V. Weaver, S. McKee, Are Cycle Accurate Simulations a Waste of Time?, in: Proc. of the 7th Workshop on Duplicating, Deconstruction and Debunking, Beijing, China, 2008.
[30] J. S. Bucy, J. Schindler, S. W. Schlosser, G. R. Ganger, The DiskSim Simulation Environment Version 4.0 Reference Manual, Tech. Rep. CMU-PDL-08-101, Carnegie Mellon University, Parallel Data Lab (2008).
[31] A. Núñez, J. Fernández, J. D. García, F. García, J. Carretero, New Techniques for Simulating High Performance MPI Applications on Large Storage Networks, Journal of Supercomputing 51 (1) (2010) 40–57.
[32] The ns-3 Network Simulator, Available at http://www.nsnam.org.
[33] K. Fujiwara, H. Casanova, Speed and Accuracy of Network Simulation in the SimGrid Framework, in: Proc. of the First International Workshop on Network Simulation Tools (NSTools), ACM, 2007.
[34] B. Penoff, A. Wagner, M. Tüxen, I. Rüngeler, MPI-NeTSim: A Network Simulation Module for MPI, in: Proc. of the 15th International Conference on Parallel and Distributed Systems (ICPADS), 2009, pp. 464–471.
[35] P. Velho, L. M. Schnorr, H. Casanova, A. Legrand, On the Validity of Flow-level TCP Network Models for Grid and Cloud Simulations, ACM Transactions on Modeling and Computer Simulation 23 (4) (2013) 23.
[36] W. Chen, E. Deelman, WorkflowSim: A Toolkit for Simulating Scientific Workflows in Distributed Environments, in: Proc. of the 8th IEEE International Conference on eScience, IEEE, 2012.
[37] F. Teng, L. Yu, F. Magoulès, SimMapReduce: A Simulator for Modeling MapReduce Framework, in: 5th FTRA International Conference on Multimedia and Ubiquitous Engineering (MUE), 2011, pp. 277 –282.
[38] Y. Shi, X. Jiang, K. Ye, An Energy-Efficient Scheme for Cloud Resource Provisioning Based on CloudSim, in: Proc. of the IEEE International Conference on Cluster Computing (CLUSTER), 2011, pp. 595 –599.
[39] J. Jung, H. Kim, MR-CloudSim: Designing and Implementing MapReduce Computing Model on CloudSim, in: International Conference on ICT Convergence (ICTC), 2012, pp. 504 –509.
[40] D. P. Bertsekas, R. G. Gallager, Data Networks, 2nd Edition, Prenctice Hall, 1996.
[41] T. Giuli, M. Baker, Narses: A Scalable Flow-Based Network Simulator, Tech. Rep. cs.PF/0211024, Stanford University, available at http://arxiv.org/abs/cs.PF/0211024 (2002).
[42] D.-M. Chiu, Some Observations on Fairness of Bandwidth Sharing, in: Proc. of the 5th IEEE Symposium on Computers and Communications (ISCC), IEEE Computer Society, Antibes, France, 2000, pp. 125–131.
[43] S. H. Low, A duality model of TCP and queue management algorithms, IEEE/ACM Transactions on Networking 11 (4) (2003) 525–536.
[44] K. Popper, Objective Knowledge: An Evolutionary Approach, Oxford University Press, 1972.
[45] M. Mathis, J. Semke, J. Mahdavi, T. Ott, The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm, Computer Communications Review 27 (3) (1997) 67–82.
[46] S. Floyd, K. Fall, Promoting the Use of End-to-end Congestion Control in the Internet, IEEE/ACM Transactions on Networking 7 (4) (1999) 458–472.
[47] M. Jain, R. S. Prasad, C. Dovrolis, The TCP Bandwidth-Delay Product Revisited: Network Buffering, Cross Traffic, and Socket Buffer Auto-Sizing, Tech. Rep. GIT-CERCS-03-02, Georgia Institute of Technology (2003).
[48] G. Marfia, C. Palazzi, G. Pau, M. Gerla, M. Sanadidi, M. Roccetti, TCP Libra: Exploring RTT-Fairness for TCP, in: Proc. of the 6th International IFIP-TC6 Networking Conference on Ad Hoc and Sensor Networks, Wireless Networks, Next Generation Internet, Vol. 4479 of LNCS, Springer, 2007, pp. 1005–1013.
[49] M. Heusse, S. A. Merritt, T. X. Brown, A. Duda, Two-way TCP Connections: Old Problem, New Insight, ACM Computer Communication Review 41 (2) (2011) 5–15.

[50] A. Tang, L. Andrew, K. Jacobsson, K. Johansson, S. Low, H. Hjalmarsson, Window Flow Control: Macroscopic Properties from Microscopic Factors, in: Proc. of the 27th IEEE International Conference on Computer Communications (INFOCOM), 2008, pp. 91–95.
[51] K. Jacobsson, L. Andrew, A. Tang, K. Johansson, H. Hjalmarsson, S. Low, ACK-Clocking Dynamics: Modelling the Interaction between Windows and the Network, in: Proc. of the 27th IEEE International Conference on Computer Communications (INFOCOM), 2008, pp. 2146–2152.
[52] A. Tang, L. Andrew, K. Jacobsson, K. Johansson, H. Hjalmarsson, S. Low, Queue Dynamics With Window Flow Control, IEEE/ACM Transactions on Networking 18 (5) (2010) 1422–1435.
[53] P. Velho, A. Legrand, Accuracy Study and Improvement of Network Simulation in the SimGrid Framework, in: Proc. of the 2nd International Conference on Simulation Tools and Techniques (SIMUTools'09), 2009.
[54] H. Casanova, L. Marchal, A Network Model for Simulation of Grid Application, Tech. Rep. 2002-40, ÉNS de Lyon, LIP (2002).
[55] E. Gabriel, G. Fagg, G. Bosilca, T. Angskun, J. Dongarra, J. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. Castain, D. Daniel, R. Graham, T. Woodall, Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation, in: Proc. of the 11th European PVM/MPI Users' Group Meeting, Vol. 3241 of Lecture Notes in Computer Science, Springer, 2004, pp. 97–104.
[56] W. Gropp, MPICH2: A new start for MPI implementations, in: Recent Advances in Parallel Virtual Machine and Message Passing Interface, 9th European PVM/MPI Users' Group Meeting, Vol. 2474 of Lecture Notes in Computer Science, Springer, 2002.
[57] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, T. von Eicken, LogP: Towards a Realistic Model of Parallel Computation, in: Proc. of the fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPOPP), San Diego, CA, 1993, pp. 1–12.
[58] A. Alexandrov, M. F. Ionescu, K. E. Schauser, C. Scheiman, LogGP: Incorporating Long Messages Into the LogP Model – One Step Closer Towards a Realistic Model for Parallel Computation, in: Proc. of the 7th ACM Symposium on Parallel Algorithms and Architectures (SPAA), Santa Barbara, CA, 1995, pp. 95–105.
[59] T. Kielmann, H. E. Bal, K. Verstoep, Fast Measurement of LogP Parameters for Message Passing Platforms, in: Proc. of the 15 IPDPS 2000 Workshops on Parallel and Distributed Processing, IPDPS '00, Springer-Verlag, London, UK, UK, 2000, pp. 1176–1183.
[60] F. Ino, N. Fujimoto, K. Hagihara, LogGPS: a Parallel Computational Model for Synchronization Analysis, in: Proc. of the eighth ACM SIGPLAN Symposium on Principles and Practices of Parallel Programming (PPoPP), Snowbird, UT, 2001, pp. 133–142.
[61] P.-N. Clauss, M. Stillwell, S. Genaud, F. Suter, H. Casanova, M. Quinson, Single Node On-Line Simulation of MPI Applications with SMPI, in: Proc. of the 25th IEEE International Parallel and Distributed Processing Symp (IPDPS), IEEE, 2011, pp. 661–672.
[62] P. Bedaride, A. Degomme, S. Genaud, A. Legrand, G. Markomanolis, M. Quinson, M. Stillwell, Lee, F. Suter, B. Videau, Toward Better Simulation of MPI Applications on Ethernet/TCP Networks, in: Proc. of the 4th International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems, Denver, United States, 2013.
[63] F. Dabek, R. Cox, M. F. Kaashoek, R. Morris, Vivaldi: A Decentralized Network Coordinate System, in: Proc. of ACM SIGCOMM, 2004, pp. 15–26.
[64] N. Hu, L. Li, Z. Mao, P. Steenkiste, J. Wang, A Measurement Study of Internet Bottlenecks, in: Proc. of the 24th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), 2005, pp. 1689–1700.
[65] M. Dischinger, A. Haeberlen, P. K. Gummadi, S. Saroiu, Characterizing Residential Broadband Networks, in: Proc. of the 7th ACM SIGCOMM Conference on Internet Measurement, ACM, 2007, pp. 43–56.
[66] O. Beaumont, L. Eyraud-Dubois, Y. J. Won, Using the Last-mile Model as a Distributed Scheme for Available Bandwidth Prediction, in: Proc. of the 17th International European Conference on Parallel and Distributed Computing, Vol. 6852 of LNCS, Springer, 2011, pp. 103–116.
[67] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, M. Bowman, PlanetLab: an Overlay Testbed for Broad-Coverage Services, ACM SIGCOMM Computer Communication Review 33 (3) (2003) 3–12.
[68] B. Donassolo, H. Casanova, A. Legrand, P. Velho, Fast and Scalable Simulation of Volunteer Computing Systems Using SimGrid, in: Proc. of the Second Workshop on Large-Scale System and Application Performance (LSAP), 2010, pp. 605–612.
[69] D. Kondo, B. Javadi, A. Iosup, D. Epema, The Failure Trace Archive: Enabling Comparative Analysis of Failures in Diverse Distributed Systems, in: Proceeding of the 10th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), IEEE, 2010, pp. 398–407.
[70] E. Heien, N. Fujimoto, K. Hagihara, Computing Low Latency Batches with Unreliable Workers in Volunteer Computing Environments, in: Proc. of the Second Workshop on Large-Scale, Volatile Desktop Grids, 2008.
[71] K. Calvert, M. Doar, E. Zegura, Modeling Internet Topology, IEEE Communications Magazine 35 (6) (1997) 160–163.
[72] B. Waxman, Routing of Multipoint Connections, IEEE Journal on Selected Areas in Communications 6 (9) (1988) 1617–1622.
[73] A. Medina, A. Lakhina, I. Matta, J. Byers, BRITE: An Approach to Universal Topology Generation, in: Proc. of the International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunications Systems (MASCOTS), 2001, pp. 346–356.
[74] L. Bobelin, A. Legrand, D. A. Gonzàlez Màrquez, P. Navarro, M. Quinson, F. Suter, C. Thiery, Scalable Multi-Purpose Network Representation for Large Scale Distributed System Simulation, in: Proceedings of the 12th IEEE International Symposium on Cluster Computing and the Grid (CCGrid'12), IEEE Computer Society Press, 2012, pp. 220–227.
[75] K. Butler, P. McDaniel, W. Aiello, Optimizing BGP security by exploiting path stability, in: Proc. of the 13th ACM Conference on Computer and Communications Security, 2006, pp. 298–310.
[76] 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, I. Touche, Grid'5000: A Large Scale And Highly Reconfigurable Experimental Grid Testbed, IJHPCA 20 (4) (2006) 481–494.
[77] M.-E. Frincu, M. Quinson, F. Suter, Handling Very Large Platforms with the New SimGrid Platform Description Formalism, Tech. Rep. 348, INRIA (2008).
[78] W. Depoorter, N. De Moor, K. Vanmechelen, J. Broeckhove, Scalability of Grid Simulators : An Evaluation, in: Proc. of the 14th EuroPar Conference, Vol. 5168 of LNCS, Springer, 2008, pp. 544–553.
[79] M. Quinson, C. Rosa, C. Thiéry, Parallel Simulation of Peer-to-Peer Systems, in: Proc. of the 12th IEEE International Symposium on Cluster Computing and the Grid (CCGrid), 2012.
[80] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. F. Kaashoek, F. Dabek, H. Balakrishnan, Chord: a Scalable Peer-to-Peer Lookup Protocol for Internet Applications, IEEE/ACM Transactions on Networking 11 (1) (2003) 17–32.
[81] J. J. Dongarra, P. Luszczek, A. Petitet, The LINPACK benchmark: Past, Present, and Future., Concurrency and Computation: Practice and Experience 15 (9) (2003) 803–820.
[82] R. S. Baker, K. R. Koch, An s_n algorithm for the massively parallel CM-200 computer, Nuclear Science and Engineering 128 (3) (1998) 312–320, available at http://wwwc3.lanl.gov/pal/software/sweep3d/.

[83] L. Genovese, A. Neelov, S. Goedecker, T. Deutsch, S. A. Ghasemi, A. Willand, D. Caliste, O. Zilberberg, M. Rayson, A. Bergman, R. Schneider, Daubechies Wavelets as a Basis Set for Density Functional Pseudopotential Calculations, Journal of Chemical Physics 129 (014109).
[84] D. Peter, D. Komatitsch, Y. Luo, R. Martin, N. Le Goff, E. Casarotti, P. Le Loher, F. Magnoni, Q. Liu, C. Blitz, T. Nissen-Meyer, P. Basini, J. Tromp, Forward and Adjoint Simulations of Seismic Wave Propagation on Fully Unstructured Hexahedral Meshes, Geophysical Journal International 186 (2) (2011) 721–739.
[85] Mont-Blanc: European Approach Towards Energy Efficient High Performance, Montblanc, http://www.montblanc-project.eu/.
[86] N. Rajovic, L. Vilanova, C. Villavieja, N. Puzovic, A. Ramirez, The Low-Power Architecture Approach Towards Exascale Computing, Journal of Computational Science 4 (6) (2013) 439–443.
[87] R. Thakur, R. Rabenseifner, W. Gropp, Optimization of Collective Communication Operations in MPICH, International Journal of High Performance Computer Applications 19 (1) (2005) 49–66.