INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Towards Scalable, Accurate, and Usable Simulations of Distributed Applications and Systems

Olivier Beaumont, Laurent Bobelin, Henri Casanova, Pierre-Nicolas Clauss, Bruno Donassolo, Lionel Eyraud-Dubois, Stéphane Genaud, Sascha Hunold, Arnaud Legrand, Martin Quinson, Cristian Rosa, Lucas Mello Schnorr, Mark Stillwell, Frédéric Suter, Christophe Thiéry, Pedro Velho, Jean-Marc Vincent, Young J. Won.

N° 7761

2011

Domaine 3

apport de recherche

ISSN 0249-6399 ISRN INRIA/RR--7761--FR+ENG

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE
INRIA
centre de recherche NANCY - GRAND EST

Towards Scalable, Accurate, and Usable Simulations of Distributed Applications and Systems

Olivier Beaumont*, Laurent Bobelin†, Henri Casanova‡, Pierre-Nicolas Clauss§, Bruno Donassolo¶, Lionel Eyraud-Dubois*, Stéphane Genaud‖, Sascha Hunold†, Arnaud Legrand†, Martin Quinson§, Cristian Rosa§, Lucas Mello Schnorr†, Mark Stillwell**, Frédéric Suter††, Christophe Thiéry§, Pedro Velho†, Jean-Marc Vincent†, Young J. Won*.

Domaine : Réseaux, systèmes et services, calcul distribué
Équipe-Projet AlGorille

Rapport de recherche n° 7761 — 2011 — 36 pages

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 algorithm and system solutions via *simulation*. Unlike direct experimentation via an application deployment on a real-world testbed, simulation enables fully repeatable and configurable experiments that can often be conducted quickly for arbitrary hypothetical scenarios. In spite of these promises, current simulation practice is often not conducive to obtaining scientifically sound results. State-of-the-art simulators are often not validated and their accuracy is unknown. Furthermore, due to the lack of accepted simulation frameworks and of transparent simulation methodologies, published simulation results are rarely reproducible. We highlight recent advances made in the context of the SIMGrid simulation framework in a view to addressing this predicament across the aforementioned domains. These advances, which pertain both to science and engineering, together lead to unprecedented combinations of simulation accuracy and scalability, allowing the user to trade off one for the other. They also enhance simulation usability and reusability so as to promote an Open Science approach for simulation-based research in the field.

Key-words: Distributed computing simulation, validation, scalability, SIMGrid

* INRIA, Bordeaux University, France
† Grenoble University, France
‡ Dept. of Computer and Information Sciences, University of Hawai'i at Manoa, U.S.A
§ Nancy University, LORIA, France
¶ Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre, Brazil
‖ University of Strasbourg, ICPS-LSIIT, Illkirch, France
** University of Lyon, LIP, INRIA, Lyon, France
†† IN2P3 Computing Center, CNRS/IN2P3, Lyon-Villeurbanne, France

Centre de recherche INRIA Nancy – Grand Est
LORIA, Technopôle de Nancy-Brabois, Campus scientifique,
615, rue du Jardin Botanique, BP 101, 54602 Villers-Lès-Nancy
Téléphone : +33 3 83 59 30 00 — Télécopie : +33 3 83 27 83 19

Contributions à l'extensibilité, la précision et l'utilisabilité des simulations de systèmes et applications distribuées

Résumé : L'étude de systèmes et applications parallèles et distribués, qu'il s'agisse de clusters, de grilles, de systèmes pair-à-pair de volunteer computing, ou de cloud, demandent souvent l'évaluation empirique par *simulation* des algorithmes et solutions proposés. Contrairement à l'expérimentation directe par déploiement d'applications sur des plates-formes réelles, la simulation permet des expériences reproductibles pouvant être menée rapidement sur n'importe quel scénario hypothétique. Malgré ces avantages théoriques, les pratiques actuelles en matière de simulation ne permettent souvent pas d'obtenir des résultats scientifiquement éprouvés. Les simulateurs classiques sont trop souvent validés et leur réalisme n'est pas démontré. De plus, le manque d'environnements de simulation communément acceptés et de méthodologies classiques de simulation font que les résultats publiés grâce à cette approche sont rarement reproductibles par la communauté. Nous présentons dans cet article les avancées récentes dans le contexte de l'environnement SIMGrid pour répondre à ces difficultés. Ces avancées, comprenant à la fois des aspects techniques et scientifiques, rendent possible une combinaison inégalée de réalisme et précision de simulation et d'extensibilité. Cela permet aux utilisateurs de choisir le grain des modèles utilisés pour ses simulations en fonction de ses besoins de réalisme et d'extensibilité. Les travaux présentés ici améliorent également l'utilisabilité et la réutilisabilité de façon à promouvoir l'approche d'Open Science pour les recherches basées sur la simulation dans notre domaine.

Mots-clés : Simulation de systèmes distribués, validation, extensibilité, 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. Due to the advent of multi-core architectures, as an answer to power and heat challenges, modern HPC systems can comprise up to hundreds of thousands of cores. Exascale systems with billions of cores are envisioned for the next decade. Platforms that aggregate multiple clusters over wide-area networks, or *grids*, have also received a lot of attention in the HPC context over the last decade with both specific software infrastructures and application deployments. Beyond HPC, distributed applications and platforms have come to prominence in the *peer-to-peer* and *volunteer computing* domains, enabled by the impressive capabilities of personal computers and high-speed personal internet connections. Leveraging this potential located at the edges of the network is an attractive proposition for many applications (e.g., content sharing, volunteer computing, data storage and retrieval, media streaming). 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 various domains, many open issues must be addressed to push current uses of these platforms further and to use them for new application domains. Relevant challenges include resource management, resource discovery and monitoring, application scheduling, data management, decentralized algorithms, energy consumption reduction, resource economics, fault-tolerance and availability, scalability and performance. Regardless of the specific context and of the research question at hand, studying and understanding the behavior of distributed applications is difficult. The goal is to assess the quality of competing algorithm and system designs with respect to precise objective metrics. Three classical approaches are used in this view: *theory*, *experimentation* and *computer simulation*. In most cases, pure theoretical analysis can be conducted at best for stringent and ultimately unrealistic assumptions regarding the underlying platforms and/or applications. Therefore, most research results are obtained using experimentation or computer simulation.

Experimentation, or the direct execution of target applications on production platforms or testbeds, seems an obvious approach for obtaining sound experimental results. Unfortunately, it often proves infeasible. Real-world platforms may not be available for the purpose of experiments, so as not to disrupt production usage. Moreover, experiments can only be conducted for the platform configurations at hand, making it difficult to explore “what if?" scenarios. Experiments may also be prohibitively time consuming, especially if large numbers of them are needed to explore many scenarios with reasonable statistical significance. Finally, conducting reproducible experiments on real-world platforms often proves difficult because of the lack of control over experimental conditions, in particular for platforms subject to changing workload and resource conditions. Even when direct experimentation is feasible and sufficient, power consumption considerations must be taken into account: for large-scale platforms such as large clusters or clouds, using the platform for extensive performance evaluation experiments can be an unacceptable expense and a waste of natural resources.

Given these difficulties, it is not surprising that many published results in the field are obtained through simulation, even though researchers often strive to obtain some

experimental results for (limited) real-world scenarios. Simulation offers an attractive alternative to experimentation because simulation experiments are *fully repeatable and configurable*. Furthermore, simulation is often less labor intensive, costly, and/or time consuming than experimentation. Consequently, simulation has been used in several areas of Computer Science for decades, e.g., for microprocessor and network protocol design. Its use for distributed computing research is less developed. In this field, although simulation holds many promises, current practices have given simulation a "bad press," for good reasons.

In computer science, simulation amounts to implementing a model, i.e., a hypothetical description of the system (e.g., equations, state automata, Petri nets, programmatic procedure). The question of how close the model is to the real world should be fundamental, especially because the model is often simplified to favor simulation speed (e.g., an analytical model based on equations is faster to evaluate than a complex event-driven procedure). Simulation can thus introduce a large bias, or accuracy loss, with respect to its real-world counterpart. Because quantifying this accuracy loss is painstaking and time-consuming, very few authors have published extensive "simulation validation" results in the literature. Consequently, countless published research results are obtained via simulation methods whose accuracy is more or less unknown.

In experimental science 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. In the field of parallel and distributed computing, however, reproducing results is rarely seen as a fundamental step (and arguably often seen as a waste of time since most authors focus on novelty at all cost). Simulation experimental methodology is rarely documented in sufficient details and the simulators, which are often ad-hoc and throw-away, are rarely made available. For instance, in 2006, [51] point out that out of 141 surveyed papers that use simulation for studying peer-to-peer systems, 30% use a custom simulator, and 50% do not even report which simulator was used. This lack of acknowledged simulation frameworks and transparent simulation methodologies means that most published simulation results are impossible to reproduce by researchers other than the authors. The irony is surely not lost on the reader here, given that simulations should be repeatable by design!

We identify three related challenges for simulation frameworks in the field of distributed computing:

1. **Accuracy** – obtaining simulation results (e.g., simulated application execution time, simulated application throughput, simulated platform utilization) that match results that would be obtained with real-world application executions, or that introduce a quantifiable bias.

2. **Scalability** – simulating large-scale applications (e.g., large number of long-running tasks, large amounts of transferred data) on large-scale platforms (e.g., large number of compute nodes, large number of network links and routers) with low time complexity (e.g., simulation times orders of magnitude shorter than simulated times) and low space complexity (e.g., simulating millions of application and platform components using only a few GBytes of RAM).

3. **Usability** – providing users with ways to instantiate simulation models, to augment or develop simulation models, to implement simulations for various application scenarios, to analyze/visualize simulation results, so as to enable repeatability of simulation results by others.

In this work we focus on SIMGrid, a unified and open simulation framework for the HPC, grid, peer-to-peer, volunteer, and cloud computing domains. More specifically, we present relevant accomplishments in the last 3 years of the SIMGrid project. SIMGrid has been referenced in survey articles about simulators of parallel and distributed applications (SPDAs), and most recently in [25]. Most of these surveys do not account for the aforementioned accomplishments, likely because these accomplishments are described across distinct conference publications. Consequently, this article summarizes all salient contributions to date, providing a unified view of how they together contribute to addressing the above challenges. Note that these contributions pertain both to "science" (e.g., novel simulation models) and "engineering" (e.g., use of efficient data structures), and together contribute to furthering the state of the art of simulation in the field.

This article is organized as follows. Section 2 discusses related work. Section 3 provides an overview of SIMGrid. Sections 4, 5 and 6 present recent developments in SIMGrid that address the interrelated challenges of accuracy, scalability and usability, respectively. Finally, Section 7 concludes with a summary of results and a broad perspective on future challenges and developments.

2 Related Work

The simulation of parallel and distributed applications has received an enormous amount of attention in the literature. Many SPDAs have been developed that employ a wide range of simulation techniques. In what follows we discuss prominent SPDAs, loosely categorized based on their objective: the simulation of distributed algorithms, of abstract applications, or of legacy applications.

2.1 Simulation of Distributed Algorithms

Many SPDAs have been designed for simulating distributed algorithms on large-scale platforms, e.g., peer-to-peer systems. These SPDAs often abstract away many systems and hardware details and rely on simple simulation models. One of the key factors that affect the algorithm performance is network latency, and it is common to evaluate algorithms based on message counts (often not accounting for network bandwidth or network contention). Simulator design is thus simplified, which makes it possible to aim for extreme scalability up to systems with several millions of peers. PeerSim [39] is likely the most widely used SPDA for theoretical peer-to-peer studies. It allows both simulation in query-cycle and discrete event modes, and was reported to scale up to one million peers when using the former mode. An advantage of PeerSim is that its simple design allows users to modify and extend it. Its main limitation is its lack of realism due to simplistic simulation models put in place for the sake of scalability. OverSim [6] attempts to address this limitation by relying on the OMNet++[68] discrete event simulation kernel. This kernel includes a packet-level network simulator comparable to NS2 [47], but also extensions for modeling compute resources. OverSim, however, does not use these extensions and only simulates communication between peers. OverSim was reported to scale up to 100,000 peers (when replacing OMNeT++ by other mechanisms), and it was never thoroughly validated against a real-world system. Such validation is arguably difficult and often not a strong priority in that community. Over the last decade, many other SPDAs have emerged, such as P2PSim [31] or PlanetSim [30]. These projects have been short-lived and are no longer maintained. It is thus difficult to

say whether more recent proposals, e.g., D-P2P-Sim [64], will perdure. The volatility of simulation technology is a clear impediment to obtaining enduring research results. As seen in Section 5, SIMGrid, which has been available and maintained for 12 years to date, achieves high enough scalability for supporting large-scale simulation of distributed algorithms while striving to maintain simulation accuracy.

2.2 Simulation of Abstract Applications

We use the term “abstract application" to denote a specification of an application as a set of possibly dependent tasks and data volumes to be exchanged between these tasks. Each task is specified by an execution cost (e.g., number of instructions, number of floating point operations, execution time on a reference host). In the end, the application implementation is completely abstracted and, in most cases, no such implementation exists. Numerous SPDAs for abstract applications have been developed in the grid computing community, most of them only intended for use by their own developers. Several were made available to the community at large, but proved to be domain-specific and short lived. For instance, ChicSim [58] and OptorSim [9] were designed to study data replication issues in grid platforms but have since been discontinued. A widely used SPDA is GridSim [14], which was initially intended for grid economy studies but has evolved to be used in a broader grid computing context. To the best of our knowledge, only two SPDAs target cloud simulation specifically. CloudSim [15] builds on the same simulation internals as GridSim but exposes specific interfaces for simulating systems that support cloud services. GroudSim [54] is a framework that enables the simulation of both grid and cloud systems. Recently, NIST has announced Koala [43], a cloud simulator that is not available at the time this article was written. Several SPDAs have been developed specifically for simulating volunteer computing systems, i.e., systems that consist of larger numbers of volatile hosts. BOINC is the most popular volunteer computing infrastructure today. It supports the concurrent execution of “projects" on large numbers of individually owned hosts, called clients. It implements various policies to determine when each client performs work units for which projects and to determine which project tasks the BOINC server should send to which clients. Several BOINC SPDAs, often simulating only a subset of its functionality, have been developed SimBA [66] models BOINC clients as finite-state automata based on availability trace files and/or probabilistic models of availability, and makes it possible to study server-side scheduling policy in simulation. The same authors later developed EmBOINC [27]. Unlike SimBA, EmBOINC instead executes actual BOINC production code to emulate the BOINC server, which makes it possible to tune the code in simulation and integrate it back into BOINC. The actual application is still completely abstract and work unit computation is implemented using simulated delays. Because of these simple client models, these SPDAs do not allow interaction between multiple servers. SimBOINC [41] goes further and simulates the full BOINC system, thus allowing for multiple servers and for the simulation of client-side scheduling. It is built with SIMGrid, and thus benefits from the features described in this work.

A common theme shared by these SPDAs is that they seek a good compromise between simulation speed and simulation accuracy. However, they often err on one side, opting for simplistic simulation models that can be computed quickly that that are unrealistic. As far as compute resource simulation, most above SPDAs use macroscopic CPU models: 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. This approach does not account for architecture-

specific features of the simulated compute resource, and is thus of questionable accuracy when simulating diverse applications on heterogeneous platforms. It is however possible to instantiate compute costs based on real-world benchmarks collected on a variety of architectures.

The simulation of storage resources is often entirely ignored. Only OptorSim and GridSim currently account for storage resources. The former merely simulates only a notion of disk capacity, which is arguably straightforward to implement as part of all the other SPDAs. GridSim does implement a disk performance model based on latency, seek time, and maximum bandwidth, but does not model any file system effects, which are known to drive disk performance. In fact, accurate analytical modeling of hard drives is an extremely challenging proposition. One alternative would then be to employ discrete event simulators of storage resources (e.g., [13]). These simulators typically model the operation of the storage hardware precisely and could in principle 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). In most SPDAs, however, such simulation is typically not attempted. An exception is the work in [53], which targets fine-grain discrete-event simulation of a storage area network. The advent of Solid State Drives (SSDs) and the possible departure from traditional hard drives may allow the development of analytical, and yet accurate, performance models of storage resources and I/O operations in the future.

The largest diversity of techniques employed in these SPDAs is for network simulation. The simplest models, used for instance in SimBA and EmBOINC, attach a bandwidth value to each host which is used to compute data transfer times for uploads or downloads given data size. A latency value may also be used, so as to obtain an affine model. Going further in sophistication, ChicSim uses a simplistic analytical model of flow data transfer rate that accounts for contention based on simple formulas. While superior to the previous model, this model does not capture the contention behavior of real-world networks. Part of the issue is that in real networks, communications are fragmented into packets. Such fragmentation is simulated in GridSim, using a simple wormhole routing strategy. Simulating fragmentation significantly increases simulation time as, by contrast with the two previous models, many simulation events are required to simulate a single network communication. In fact, the simulation time could become as large as that seen when using accurate packet-level simulators used in the network research community, e.g., NS2 [47]. Unfortunately, arbitrary packetization and wormhole routing does not model network protocol effects such as TCP flow management, and could thus be a poor alternative to full-fledge packet-level simulation. Another option is to use analytical network models that are designed to represent data transfer rate allocation as achieved by real network protocols such as TCP. The goal is to be orders of magnitude faster than packet-level simulation, while remaining close to the behavior of real-world networks. This approach is seen in OptorSim, GroudSim, and SIMGrid. In these works, communications are represented as flows in pipes, and data transfer rates are obtained with respect to a bandwidth sharing model [46]. Unfortunately, designing accurate and correct such models is challenging. For instance, OptorSim uses a flawed model that is acknowledged in the simulator's documentation but not resolved at the time this article is being written. Essentially, the bandwidth share that each flow receives on a congested network link depends only on the number of flows using this link. This computation, however, does not take into account the fact that some of these flows may be limited by other links in their path, leading to wasted bandwidth in various links. In real networks this wasted bandwidth would instead be shared between other (not otherwise limited) flows. Similar issues are found in other simulators, e.g., GroudSim.

By contrast, as seen in Section 4, SIMGrid implements scalable and accurate (i.e., experimentally validated) analytical network simulation models.

2.3 Simulation of Legacy Applications

Rather than simulating abstractions of applications, another option is to simulate actual implementations. This approach has been pursued actively in the context of parallel message-passing applications implemented with MPI [33]. Two 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-world platform. A SPDA then replays the execution of the application as if it were running on another platform with different hardware characteristics. This approach is very popular, as shown by the number of off-line SPDAs described in the literature since as recently as 2009 [36, 37, 53, 67, 72]. Off-line simulation face the difficulty that event logs can be large, requiring creative solutions as seen for instance in the PSINS [67] or PHANTOM [72] projects. Furthermore, event logs are tied to particular application execution (e.g., number of processors, block size, data distribution schemes) so that a new log must be obtained for each potential execution scenario, although extrapolation may be feasible, as seen in [37, 53]. A way to side-step these difficulties altogether is on-line simulation, in which actual application code, with no or marginal modifications, 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 SPDA. LAPSE is a well-known on-line SPDA developed in the early 90's [23]. In LAPSE, the parallel application executes normally but communication delays are injected based on a simple model of a hypothetical network. MPI-SIM [3] and the project in [60] add I/O subsystem simulation in addition to network simulation. The BigSim project [73], unlike MPI-SIM, allows the simulation of arbitrary computational delays on the target platform. Note that extrapolating computation times measured on the host platform to computation time to a platform with a different computer architecture is in general not possible, thus precluding the accurate simulation of a heterogeneous platform. A way to address this limitation, used in [44], is to use a cycle-accurate hardware simulator for determining computation delays, which leads to a high ratio of simulation time to simulated time.

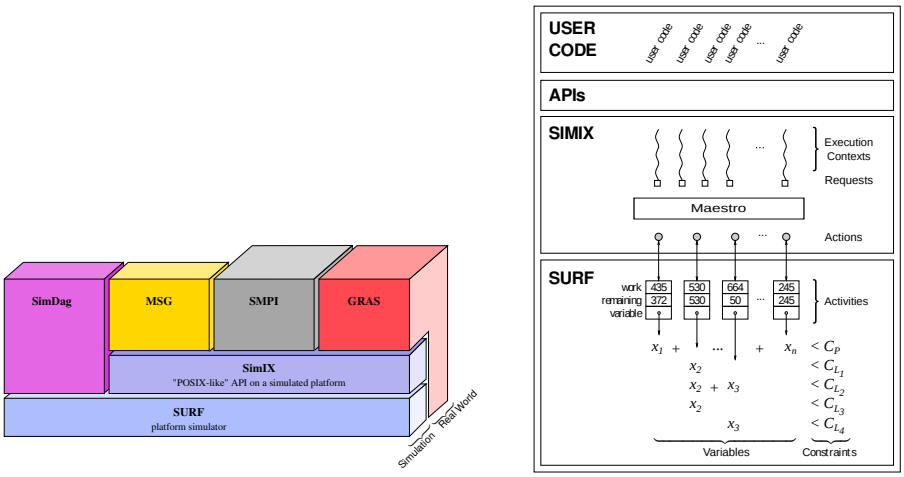
A challenge faced by all above SPDAs is the simulation of network communication. Using a packet-level network simulation, as done in MPI-NetSim [55], is ultimately unscalable. Instead, as seen in the case of SPDAs for abstract applications, most authors adopt simplistic network models. These models typically ignore network contention and use monolithic performance models for collective communications. Some authors have attempted to capture contention by using probability distributions of communication times [34]. SIMGrid can be used as a foundation for implementing off-line simulation of legacy application, but also supports on-line simulation. Regardless, it enables fast and scalable simulation of legacy applications via the use of accurate (i.e., experimentally validated) analytical network models. For scalability reasons, many of the simulators cited in this section run the simulation itself on a cluster platform. Instead, as seen in Section 5.4, SIMGrid makes it possible to run simulations at scale on a single computer.

3 The SIMGrid Framework

SIMGrid is a 12-year old open source project whose domain of application has kept growing since its inception. It was initiated in 1999 as a tool for studying scheduling

algorithms for heterogeneous platforms. SIMGrid v1 [17] made it easy to prototype scheduling heuristics and to test them on a variety of abstract applications (expressed as task graphs) and platforms. In 2003, SIMGrid v2 [18] extended the capabilities of its predecessor in two major ways. First, the accuracy of the simulation models was improved by transitioning the network model from a wormhole model to an analytical fluid model. Second, an API was added to simulate generic Concurrent Sequential Processes (CSP) scenarios. SIMGrid v3.0 was released in 2005, but major new features appeared in version v3.3 in April 2009. These features include a complete rewrite of the simulation core for better modularity, speed and scalability; the possibility to attach traces to resources to simulate time-dependent performance characteristics as well as failure events; and two new user interfaces.

The current software stack with its relevant components is depicted in Figure 1(a). The four components on the top of the figure, SimDag, MSG, SMPI, and GRAS are *user interfaces*. The two components below, SimIX and SURF, form the *simulation core*. A last component, not shown in the figure but used throughout the software stack up to user-space, is a general-purpose toolbox that implements classical data containers (e.g., FIFO, dynamic arrays, hash maps), logging and exception mechanisms, and support mechanisms for configuration and portability. The following two sections describe the user interfaces and the simulation core in detail.



(a) SIMGrid components (b) Implementation of a SIMGrid simulation

Figure 1: The SIMGrid software

3.1 User Interfaces

SIMGrid provides four application programming interfaces (APIs). Two of these APIs are designed for simulating the execution of applications based on an abstract specification of the application. The **SimDag** API allows the simulation of parallel applications structured as directed acyclic graphs (DAGs). Vertices denote (sequential or parallel) tasks and edges denote task dependencies and optional data transfers between tasks. A large literature is devoted to the study of DAG scheduling algorithms, and SimDag provides the necessary abstractions to quickly implement and evaluate such algorithms. The **MSG** API is intended for CSP simulation and provides classic CSP

abstractions (processes, mailboxes, channels, etc.). It is therefore generic and, to date, it is the most widely used SIMGrid API, providing bindings for C, Java, Ruby and Lua.

The remaining two APIs are designed for simulating the execution of applications based on actual application source code. The **SMPI** API [22] targets the on-line simulation of MPI applications. Actual application code is executed and MPI calls are intercepted so that communication delays can be injected based on network simulation. The **GRAS** API [56] makes it possible to use SIMGrid as a development framework for implementing full-fledged distributed applications. It provides two back-ends, allowing the same application source code to be executed either in simulation or deployed on a real-world platform. GRAS can thus bypass the simulation core entirely, as depicted in Figure 1(a). Consequently, application developers benefit from an enhanced development cycle in which the application can be quickly tested over arbitrary simulation scenarios as it is being developed.

3.2 Simulation Core

3.2.1 SURF and SimIX

The component that implements all simulation models available in SIMGrid is called **SURF**. It provides an abstract interface to these models that exposes them as *resources* (i.e. network links, workstations) and *activities* that can consume these resources. For convenience, an additional layer is provided, called **SimIX**. It provides POSIX-like services including *processes*, IPC, locks, and *actions*. SimIX processes correspond to execution contexts (e.g., threads) for the simulated application, that run code written by the SIMGrid user using one of the provided APIs. All APIs but SimDag are written using SimIX. This exception is because SimDag only allows the user to simulate centralized algorithms without independent processes, removing the need for the SimIX layer.

SimIX acts as a virtual operating system that provides a system call interface through which processes place *requests*. These requests are used for all interaction between the user program and the simulated platform. SimIX actions connect the requests from the user programs (expressed through the APIs) and the activities on the simulated resources in SURF. These activities are used by SURF to compute the delays incurred by the user actions (e.g., computations and communication operations). Each activity is represented by a data structure that stores the total amount of "work" to be done (e.g., number of bytes to transfer, number of compute operations to perform) and the amount of work remaining. A process blocks on a request until it is answered when the delay corresponding to all the activities currently performed by the process have expired. Simply put, if a process issues a request that would initiate an activity that should take x time units, this process blocks on the request until SimIX answers it, i.e., once the simulated clock has advanced by x time units. This scheme is depicted in Figure 1(b). For instance, the third activity depicted in the figure corresponds to a total amount of work of 664 units, and 50 of these units remain until completion of the activity. When the activity completes, the request depicted above the action associated to the activity is answered, allowing the corresponding process to continue execution. SIMGrid is designed so that the simulation state can only be updated in the SimIX layer. Furthermore, as depicted in the figure, updates are all performed by a unique execution context, e.g., a thread, which we call the *maestro*. When a process places a request to the maestro it is blocked until the maestro has serviced the request. The rational for using a maestro is explained in Section 5.2.

Algorithm 1 Main simulation loop

```
1: readyset ← all processes
2: while readyset ≠ ∅ do
3:    requests ← run_processes(readyset)
4:    handle_requests(requests)
5:    (t1, activities) ← compute_next_activity_completions()
6:    t2 ← compute_next_resource_state_change()
7:    t ← min(t1, t2)
8:    update_simulation_state(t)
9:    readyset ← answer_requests(activities)
10: end while
```

3.2.2 The Main Simulation Loop

SimIX implements a "simulation loop" through which the simulation makes progress, as shown in pseudo-code in Algorithm 1. The algorithm maintains a set of ready processes, i.e., new processes or those processes whose requests have been answered. The loop executes the processes in the ready set each in turn, and ends when there are no such processes in the ready set (which is either the end of the simulation or a detected deadlock). At the beginning of each iteration, SimIX lets all ready processes execute (line 3). By default, all processes are run in mutual exclusion and in round-robin fashion. Each process runs until completion or until it issues a request. Next, the SimIX maestro handles the set of requests issued by the processes, possibly creating or canceling activities (line 4). These requests are processed in deterministic order based on process IDs to ensure simulation repeatability. For each simulated resource with at least one pending activity, SURF determines when each pending activity will complete (note that an activity may use more than one resource). The minimum of these completion dates is then computed and the set of activities that complete at that minimum date is determined (line 5). SIMGrid allows users to attach "traces" to simulated resources. These traces are time-stamped lists of resource states. They are used for instance to simulate time-dependent resource availability for an out-of-band workload that causes fluctuations in the performance delivered by the resources. SURF computes the earliest resource state change date (line 6), and then the minimum of this date and of the minimum activity completion date (line 7). The state of the simulation is then advanced to this date, updating activity states and resource states (line 8). Finally, based on those activities that have completed, the corresponding requests are answered thus unblocking the relevant processes and updating the ready set(line 9).

3.2.3 Simulation Model Formalization and Implementation

During the main simulation loop, SURF determines execution rates and completion times of activities on resources. This determination is based on the various simulation models implemented in SIMGrid, several of which are discussed in upcoming sections. It turns out that most of these models can be formalized in a unified manner as a multi-variate optimization problem subject to linear constraints. As depicted at the bottom of Figure 1(b), a variable is associated to each activity that quantifies the activity's execution rate. SIMGrid implements an efficient sparse representation of the set of linear constraints, and solves the optimization problem with time complexity linear in the number of activity variables and the number of resources. In principle this computation

has to be performed at each iteration of the main simulation loop. However, Section 5.1 describes ways to reduce the involved computational cost.

4 Simulation Accuracy

The simulation of parallel and distributed applications typically entails simulating three types of resources and of activities on these resources: (i) storage resources and I/O operations; (ii) compute resources and computations; and (iii) network resources and data transfers. For each resource type, as seen in Section 2, a common theme is found: the alternative to simple, quick-to-compute, but inaccurate models, if any, is the use of fine-grain discrete-event techniques acknowledged to be accurate but leading to simulation times prohibitively high for a vast number of relevant use cases. Bridging the gap between these two extremes, whenever possible, is one of the main objectives of the SIMGrid project. To date, major advances have been made on the network simulation front, i.e., the development of analytical network models that are drastically more accurate than state-of-the-art analytical models used by SPDAs. Relevant contributions are highlighted in the next two sections.

4.1 TCP Network Models in WANs

To the best of our knowledge, SIMGrid was the first SPDA to propose and implement an analytical network model that goes beyond the naïve latency and bandwidth affine models or the wormhole routing models described in Section 2. More specifically, SIMGrid implements a *flow model*, by which the bandwidth allocated to each network flow is computed based on the underlying network topology and a bandwidth sharing model. The goal is to use a model that closely approximates the bandwidth sharing that emerges from the use of standard network protocols such as TCP. A popular and simple bandwidth sharing model is Max-Min fairness [10], by which the bandwidth allocation is such that increasing the allocation of a flow would necessarily require decreasing the allocation of another. Unfortunately, it is known that TCP does not implement Max-Min fairness [20], in part because flow bandwidth is limited by the TCP congestion window and the flow's round-trip time (RTT). Based on these considerations, the following flow model was initially implemented in SIMGrid [19]:

$$\begin{array}{l} \text{MAXIMIZE } \min_i RTT_i.\rho_i, \\ \text{UNDER CONSTRAINTS} \\ \begin{cases} \forall \mathcal{L}_k, \sum_{i|\mathcal{F}_i \text{ uses } \mathcal{L}_k} \rho_i \leqslant B_k \\ \forall \mathcal{F}_i, \rho_i \leqslant \frac{W}{RTT_i} \end{cases} \end{array} \tag{1}$$

where $\mathcal{L}_k$ denotes a network link with bandwidth capacity B_k, $\mathcal{F}_i$ denotes a network flow with assigned bandwidth ρ_i and RTT RTT_i, and W denotes the TCP congestion window size. This model is superior to the aforementioned simpler models (as easily demonstrated with simple network topologies and flow sets), and can be computed orders of magnitude faster than full-fledge packet-level simulation. SIMGrid also provides a transparent interface to the GTNetS [61] and ns-3 [52] packet-level simulators, intended for users who can tolerate the loss of simulation speed for the benefit of more realistic network simulation. It is thus straightforward to use SIMGrid to compare the above model to packet-level simulation, and a first such comparison is presented in [28]. Results therein revealed two limitations of the model: (i) unlike in packet-level

simulation TCP's slow-start behavior is not captured, which makes the model inaccurate for data transfer sizes under around 10MBytes; and (ii) bandwidth sharing becomes inaccurate in highly congested scenarios.

In [69], we conducted a new set of validation experiments in a view to identifying and remedying the above limitations. There is no hope for a flow model to capture TCP's slow start behavior perfectly. However, it turns out that an empirical model can be derived that gives a more accurate expression for the data transfer time T for a flow with round-trip time RTT and congestion window W, which is is allocated bandwidth B by the bandwidth sharing model. In the original model, the expression for T was:

$$T^{\text{original}} = L + \frac{S}{\min(B, \frac{W}{RTT})} \ . \tag{2}$$

The new model, with two additional parameters computed empirically to minimize model error with respect to packet-level simulation results obtained on a set of benchmark configurations, is:

$$T^{\text{improved}} = 10.4 \times L + \frac{S}{\min(0.92 \times B, \frac{W}{RTT})} \ . \tag{3}$$

Comparisons with GTNetS show that the new model is accurate for data sizes as low as 100KBytes, i.e., about two orders of magnitude smaller than previously achieved. For data sizes smaller than 100KBytes, the flow model produces shorter transfer times than what would happen in a real network. We suspect that below this limit, the assumption that the transfer time is as a linear function of the data size breaks down because data gets exchanged as discrete network packets. (This effect is crucial for simulating HPC application over LAN networks, and Section 4.2 presents a solution in such settings.) At any rate, users wanting to simulate small-size data transfers over wide area networks are reduced to two options: either configure SIMGrid to use costly packet-level simulation or account for optimistic simulated transfer times when drawing conclusions from simulation results.

Packet-level experiments with a dumbbell network topology have shown that as network contention increases, bandwidth sharing is no longer solely affected by flow RTTs but also by physical link bandwidths. These results lead to the following proposal for an improved bandwidth sharing model:

$$\begin{array}{l} \text{MAXIMIZE } \min_i w_i.\rho_i, \\ \text{UNDER CONSTRAINTS} \\ \begin{cases} \forall \mathcal{L}_k, \sum_{i|\mathcal{F}_i \text{ uses } \mathcal{L}_k} \rho_i \leqslant 0.92 \times B_k \\ \forall \mathcal{F}_i, \rho_i \leqslant \frac{W}{w_i} \\ w_i = \sum_{k|\mathcal{F}_i \text{ uses } \mathcal{L}_k} (L_k + \frac{\sigma}{B_k}) \end{cases} \end{array} \tag{4}$$

where σ is a constant to be determined. Note the 0.92 factor that bounds the achievable bandwidth on a link. It turns out that this factor, which was determined experimentally, corresponds to the communication's payload, i.e., the fraction of the transferred bytes that contain actual data. The effect of latency on bandwidth sharing is capture by the additional σ/B_k terms. For a set of benchmark experiments, we empirically found that $\sigma = 8775.0$ minimizes model error.

While the above model holds for simple network topologies, one may wonder how it fares in general, in particular because it is instantiated based on a finite set of benchmark experiments. To answer this questions we generated random network topologies with 50

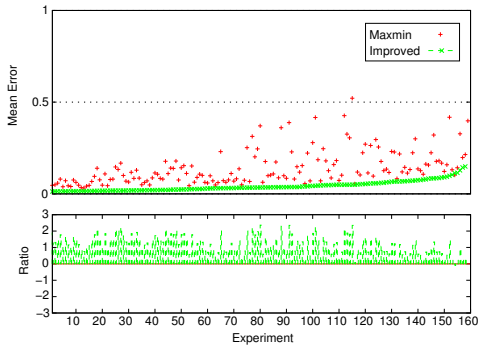


Figure 2: Simulation error comparison between the original and the improved network flow model used in SIMGrid.

and 200 nodes, using both the Waxman model [70] and the BRITE generator [48], with various latency and bandwidth distributions. In each experiment, such a topology is selected and 150 flows are created each between two random end-points. Figure 4 shows a set of results from 160 such experiments. It depicts, for each experiment, the mean error (with respect to packet-level simulation) with the original MaxMin model and with the improved model. The top graph shows actual error while the bottom graph shows error ratio. The conclusion, supported by other results in [69], is that this improved analytical network model is much close to the accuracy of packet-level simulation, thus further bridging the gap between the two approaches. The improved model has now replaced the original model in the current SIMGrid version.

4.2 TCP Network Models in Clusters

Many SPDA users wish to simulate application execution on cluster platforms, targeting either abstract or legacy applications from the HPC domain. As seen in Section 3, SIMGrid provides interfaces for both types of simulations. Regardless of the type of simulation, there is a need for an accurate network model that is representative of cluster interconnects. The model presented in the previous section proves accurate, within certain limits, for simulating network flows on a WAN. However, it fails to capture some of the fundamental aspects of the behavior of real-world cluster interconnects using TCP and popular MPI implementations, e.g., OpenMPI [29] or MPICH2 [32] over a Gigabit Ethernet switch. 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. More importantly, implementations for `MPI_Send` typically switch from buffered to synchronous mode above a certain message size. The former case involves an extra data copy performed by the MPI implementation, with the goal to avoid copying large amounts of data. This feature is seen both in OpenMPI or MPICH2. The combination of such effects is that instead of being an affine function of message size, as in Eq 3, communication time is piece-wise linear. The difference between affine and piece-wise linear in terms of accuracy proves to be large enough to cause large inaccuracies when simulating applications that perform many communications on clusters. In [22] we have described an enhancement to the network model in SIMGrid presented in the previous section. This enhancement makes it possible to model communication time with a piece-wise linear model using an arbitrary number of linear segments.

One challenge when using this model is its instantiation. Based on experiments conducted on several production clusters, we found that it is sufficient to use 3 linear segments in practice, which at first glance requires 8 model parameters (2 for defining the boundaries of the 3 segments, and one latency and bandwidth parameter for each segment). Some of these parameters, however, depend on each other thus making the model fully defined by 6 parameters. At any rate, the number of parameters is much higher than when using the simplistic affine model. We opt for an empirical approach, by which the model is instantiated based on a set of real measurements using linear regression. The number of segments and the segment boundaries are chosen such that the product of the correlation coefficients between the model and the actual measurements is maximized.

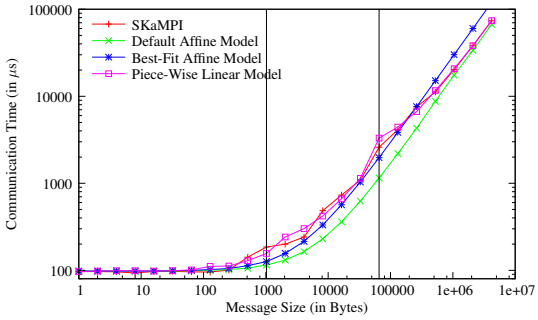


Figure 3: Comparison between a real execution (using SKaMPI over OpenMPI) and simulation models (default affine, best-fit affine, and piece-wise linear models) for a ping-pong communication between two nodes of the cluster used for instantiating the simulation models.

Figure 3 presents results obtained with the ping-pong benchmark provided as part of the SKaMPI [59] MPI benchmarking framework. The figure plots benchmark times vs. message size in bytes, using a logarithmic scale for both axes. Results are presented for a real execution of the benchmark on a production cluster, and for simulation of the benchmark using three different simulation models. These simulation models were instantiated based on experiments conducted on the same cluster. The “Default Affine" simulation model correspond to the standard model used in most state-of-the-art SPDAs that use only two parameters: latency and bandwidth. The latency is instantiated based on the time to send a 1-byte message on the cluster, and the bandwidth based on the maximum achievable bandwidth using TCP/IP, which is approximately 92% of the peak bandwidth, as seen in the previous section. The “Best-Fit Affine" model, which cannot be instantiated in practice in general, corresponds to those choices for the latency and bandwidth parameters that minimize model error with respect to the SKaMPI results. Finally, the “Piece-wise Linear" model is the model described earlier, whose segments are depicted in the figure via vertical lines.

The main observation is that the piece-wise linear model matches the real-world results well (at most a 8.63% average error overall, with worst error at 27%). By contrast, both affine models fail to capture the entire real-world behavior. The Default Affine model is accurate for small and big messages, but inaccurate in between (for a 32.1% average error overall, with worst case at 127%). The Best-Fit Affine model performs better for medium-sized messages, but overestimates communication time for big messages (for a 18.5% average error overall, with worst case at 62.6%). An

affine model is thus inherently less accurate than a piece-wise model. This is expected, but it is important to note that the decrease in accuracy is large. Further results in [22] 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 this approach remains reasonably accurate when applied to a sequence of switches. This is an important point because large compute clusters are often organized as networks of switches.

The piece-wise linear model of point-to-point communication described in this section, when combined with the bandwidth sharing model described in the previous section, leads to an immediate simulation model for collective communication operations. Just like in any MPI implementation, collective communications are implemented in SMPI as sets of point-to-point communications that may experience network contention among themselves. This is to be contrasted with monolithic modeling of collective communications, as done in [67] 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. Results in [22] show that SMPI simulates collective communications effectively. For instance, an all-to-all operation for 4MBytes of data involving 16 MPI processes is shown to be simulated with less than 1% relative error when compared to OpenMPI, which is comparable to the relative difference between OpenMPI and MPICH2.

5 Simulation Scalability

As described in Section 2, some SPDAs are not necessarily designed with scalability in mind while others have aggressively pursued it at the expense of accuracy. One objective of SIMGrid is to afford good simulation scalability across a range of domains, including simulation of peer-to-peer applications and of volunteer computing applications, while achieving simulation accuracy superior to that achieved by state-of-the-art simulators in these domains. In this section we highlight recent advances that each enhance the scalability of SIMGrid simulations using different but complementary approaches.

5.1 Optimized Simulation Loop

The main simulation loop described in Section 3.2 turns out to hinder simulation scalability. In [26] we have proposed two simple modifications that improve scalability by orders of magnitude for entire classes of simulation scenarios: lazy activity updates and trace integration.

SIMGrid was first intended for the simulation of abstract applications with many communicating tasks on nodes connected by hierarchical networks. In this setting most events regarding simulated activities and resources (e.g., starting a new activity, completion of an activity, resource status change) can have an impact on most simulated activities and resources. For instance, when executing a tightly coupled application on compute nodes interconnected via a hierarchical network, the completion of one communication can immediately impact the data transfer rates achieved by all other pending communications. Consequently, the compute_next_activity_completions (Algorithm 1, line 5) and update_simulation_state (Algorithm 1, line 8) functions loop over all activities, recomputing completion dates and updating remaining work amounts. However,

when simulating large-scale platforms such as those used for peer-to-peer or volunteer computing applications, many activities are independent of each other. For such simulations, the above functions prove to be scalability bottlenecks because they always consider all activities while many could simply be ignored most of the time. Our first proposed improvement consists in avoiding having compute_next_activity_completions recompute all activity completion dates at every iteration of the simulation loop. Instead, activities are stored in a heap based on their completion dates. When a resource share is modified, all *corresponding* activities are removed from this heap in update_simulation_state, their completion dates are updated and, and they are re-inserted into the heap, while other activities are simply not considered. update_simulation_state also removes completed activities from the aforementioned heap. Removing and inserting into a heap has time complexity $O(\log n)$, where n is the number of activities. Importantly, because retrieving the minimum element from a heap has complexity $O(1)$, compute_next_activity_completions can compute the minimum completion date and retrieve the completed tasks at that date in constant time. Note that in a simulation in which most activity completion dates need to be updated at each iteration, the above "lazy update" scheme would slightly increase time complexity compared to the original implementation. For this reason, lazy updates can be deactivated by the user.

Our second scalability improvement targets the management of changing resource states (Algorithm 1, lines 6 and 7). If t_1 is larger than t_2, then multiple resource state changes may occur before any activity completes. For instance, let us consider a situation in which the next 100 resource state changes pertain to fluctuating performance levels as specified in a user-provided trace. Furthermore, let us assume that pending activities still have large remaining work amounts so that the earliest activity completion occurs after the 100th resource state change. In this case, it is possible to aggregate 100 iterations of the main loop into one iteration. More formally, given current remaining work amounts, one can compute the next activity completion date given all future resource states before this date. This computation can be performed efficiently using "trace integration." Essentially, instead of storing a trace as performance rate values, one stores its integral. Finding the last resource state change before the next activity completion can then be performed using a binary search, i.e., with logarithmic time complexity.

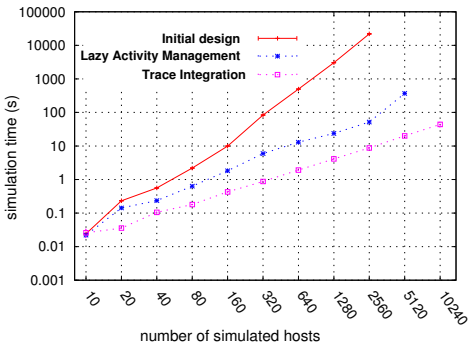


Figure 4: Simulation scalability for a volunteer computing simulation using the initial SURF design, when adding lazy activity management, and when adding trace integration.

To demonstrate the benefit afforded by these two modifications, 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 completion of the simulation. With the original design, the time complexity of this simulation is $O(N^2(P+T))$. With lazy activity management 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 [42]. 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 [35] to highlight scalability issues in previous versions of SIMGrid. Figure 4 shows simulation time measured on a 2.2GHz AMD Opteron processor vs. N for the initial design, the addition of of lazy activity management, and the addition of trace integration, using a logarithmic scale on the vertical axis. Results make it plain that both proposed improvements increase simulation scalability significantly. For instance, bounding simulation time to 10 seconds, the initial design scales up to 160 hosts. The use of lazy activity management scales up to over 600 hosts. The use of trace integration pushes this number beyond 2,600 hosts, or a more than 16-fold improvement in scalability when compared to the original design. A comparison with the state-of-the-art SimBA simulator [66], 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. This is an impressive result given that the simulation with SIMGrid is more accurate than that with SimBA (e.g., the network and the behavior of the clients are simulated).

5.2 Simulation Core Parallelization to Support Millions of Simulated Contexts

The enhancements described in the previous section may not be sufficient to achieve desired levels of scalability, especially for scaling up to millions of nodes as often needed for realistic simulation of peer-to-peer systems. In the end, scaling up the simulation simply requires "throwing more hardware resources at it." SIMGrid has been implemented in a memory-conscious manner but a given simulation has an inherent memory footprint that mandates a certain memory capacity. Scaling up a memory-bound simulation requires adding physical memory capacity. If the simulation is CPU-bound, then scaling the simulation can be achieved by exploiting multiple cores. Unfortunately, the main simulation loop (Algorithm 1) is sequential. We present hereafter our first attempt at parallelizing this loop so that SIMGrid simulations can take advantage of multi-core processors.

We first need to identify the parallelization opportunities in Algorithm 1. As explained in Section 5.1, the functions invoked at lines 5 and 6 have been re-engineered to achieve logarithmic complexity for large-scale simulation. Similarly, the handle_requests and answer_requests functions invoked at line 4 and 9 amounts for a small fraction of the simulation time. Parallelizing these functions would thus not lead to a significant speedup. Parallelizing the update_simulation_state function invoked at line 8 would entail parallelizing the resolution of the constrained optimization problem described in Section 3.2.3. Since this parallelization is possible but challenging, we have focused our efforts on the run_processes function instead, which does provide clear parallelization opportunities. Recall that this function executes user code or, more

precisely, user code fragments in between calls to the SimIX layer via user-level APIs. with SimIX, updates to the (shared) simulation states are sequentialized in a single thread, called the maestro (see Section 3.2.1). Consequently, it is possible to execute user code fragments concurrently without facing concurrency issues that arise when sharing memory among multiple threads (mutual exclusion, race conditions, deadlocks).

An engineering challenge here is the choice of the technology for implementing user processes. While standard threads could be employed, hard limits on their number imposed by operating systems (in the order of thousands) would preclude running the millions of processes needed for simulating large-scale peer-to-peer systems. Even if this limit is not reached, thread context-switching overhead would likely be prohibitive. Many existing SPDAs use standard threads, e.g., [14], and thus suffer from these scalability limitations. Instead, we implement each user process as a ucontext, as provided as part of the POSIX standard. While ucontexts were initially designed as evolutions of `setjmp`/`longjmp` functions, we can use them to execute user processes. We then run multiple ucontexts in a thread pool, since running multiple threads makes it possible to utilize multiple cores. The threads in the thread pool are synchronized with a barrier, which can be implemented easily using thread synchronization primitives. On Linux systems, we have implemented a more efficient barrier with a combination of operating system primitives and hardware atomic operations such as Compare-And-Swap.

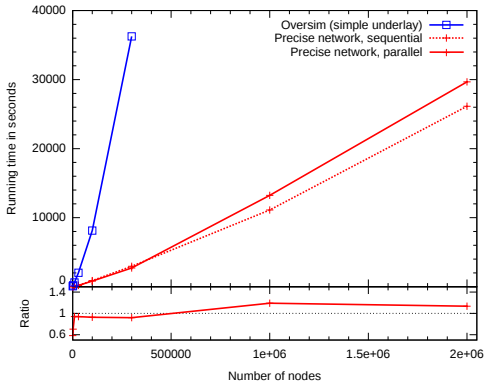


Figure 5: Simulation scalability for a Chord simulation using OverSim, the sequential implementation of run_process, and the parallel implementation of run_process.

The main question is whether the benefit of our parallelization offsets its overhead, i.e., the use of a single maestro and the barrier synchronization at each iteration. If user processes spend a large amount of time in user code, parallelization should be beneficial. If instead the simulation consists of processes that do little other than placing SimIX calls via API calls, then the overhead likely dominates. To answer this question in a practical context, we have implemented a simulation of the Chord [65] Distributed Hash Table protocol. This famous protocol was designed for scalable content indexing and retrieval on peer-to-peer platforms with significant churn. Peers are organized in a logical ring with additional cross-cutting connections, using routing tables with a number of entries logarithmic in the total number of peers. Chord is representative of a large body of algorithms in the peer-to-peer community, which is why we chose it for our simulation. Also, it is a difficult test for parallelization since user processes perform little computation in between calls to SimIX to implement the Chord protocol.

We perform an experiment similar to that reported in [6] with the OverSim simulator, on a machine with two 12-core 1.7GHz AMD CPUs and 48GB of RAM. Three sets of results are presented in Figure 5, showing simulation time vs. the number of simulated peers. The first set is obtained with OverSim using a simple constant delay model. The second set is obtained with SIMGrid, using the realistic model described in Section 4.1 and the sequential implementation of run_processes. The this set is obtained using the parallelized implementation of run_processes. The first observation is that SIMGrid affords dramatically better scalability than OverSim. For instance, SIMGrid makes it possible to simulate up to 2 millions of peers in about 8 hours using a realistic model, while OverSim simulates only 250,000 in the same amount of time. The second observation is that the parallelization of run_processes brings some benefit. The bottom part of the graph plots the parallel speedup, which reaches values above 1.2. While this is poor parallel efficiency given the total number of cores, it nevertheless represents an appreciable time saving for the user. Expectedly, other results obtained when using SIMGrid with a fixed delay model, identical to that used by OverSim, shows a higher speedup (up to 40%). Recall that Chord is a difficult test case for our parallelization because user processes perform very little computation in between calls to SimIX. Higher speedups are thus to be expected for many relevant simulated applications.

5.3 Highly Scalable “Last Mile" Network Model

One of the goals of SIMGrid is to provide users with a range of simulation models for the network that each corresponds to a different compromise between accuracy and scalability. The highest accuracy is achieved via packet-level simulation (with GTNeTS or ns-3), which is also the least scalable method. The default network model, with enhancements described in Section 4, leads to comparable albeit reduced accuracy, and is orders of magnitude more scalable. Finally, in the previous section, we have mentioned the use of a fixed delay network model. This model is not accurate, but because it provides high scalability, it is often used in the peer-to-peer community. The gap between the default model and the fixed delay model in terms of scalability is significant. For instance, when using the default model for a realistic internet topology, the scalability of the Chord simulation presented in the previous section is drastically reduced. While with the fixed delay model one can simulate 2 million peers in under 6 hours, with the default model fewer than 300,000 peers can be simulated in that amount of time. Note that this is still 50% more than what can be done with OverSim, which uses the fixed delay model. One interesting question is whether there is a model that provides yet another accuracy/scalability trade-off that falls between the fixed delay and the default model.

An approach for developing an intermediate model is to account for bandwidth at the edge of the network, while ignoring network topology. This would represent a compromise between the fixed delay model, which does not model bandwidth effects, and the default model, which models a complex topology with multi-hop network paths. Previous research on the properties of the Internet 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. [38] 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 [24]. In [7] we have analyzed a dataset obtained on the PlanetLab platform [21], and also found that most end-to-end paths seem to be limited by bandwidth at the end-points. 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.

In [7] we have developed a decentralized algorithm that can be used to instantiate the last mile 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, we have shown that this model achieves good accuracy using a small number of measurements (each host only performs bandwidth measurements with 16 other hosts). This model is thus simple and instantiable, leads to reasonable results for simulating wide-area network topologies, and is dramatically more accurate than the fixed delay model. It is also scalable since the bandwidth assignment for a communication can be computed in constant time. This last mile model was unified with the network model described in Section 4.1, without requiring a separate implementation in SURF. It is thus now possible to integrate in the same simulated platform both accurate descriptions of interconnection networks such as the ones used in grids or clusters and less detailed ones using the last mile model, which are well-suited to peer-to-peer and volunteer computing scenarios. This flexibility could be invaluable for simulating cloud computing platforms, for instance, which can combine both types of scenarios.

5.4 Scalable On-line Simulation of Legacy HPC Applications

SMPI allows SIMGrid users to conduct on-line simulations of legacy parallel applications implemented with MPI. As reviewed in Section 2.3, several SPDAs have been proposed and developed with this objective. The advantage of SMPI is that it builds on the advances made in the SIMGrid project, many of which are highlighted in this article. One well known challenge for on-line simulation of legacy parallel applications is, however, scalability. Most SPDAs reviewed in Section 2.3 address this challenge by running the simulation on a cluster. Essentially, the simulated application executes at scale, but certain operations are intercepted by a simulation component that injects a simulated delay into the application execution. For instance, MPI-NetSim [55] executes the application on a cluster, and uses one additional compute node to perform packet-level simulation for computing simulated communication delays. The scale of the simulation is thus limited by the scale of platforms at hand, thereby precluding large-scale simulations for the majority of users. Instead, via SMPI, SIMGrid attempts to run such simulations on a single computer, i.e., with orders of magnitude less CPU power and RAM capacity than needed for an actual application execution. This may seem an unachievable goal given that actual application code must be executed. However, as described hereafter, SMPI implements two techniques by which the application code is modified to achieve scalable simulation on a single computer, which we call the *host*. In the current release of SIMGrid, some of these modifications are handled automatically by a preprocessor, i.e., `smpicc`, while some still require user intervention to insert macros in the application source code. An ongoing development goal of SMPI is to automate this process entirely.

Reducing CPU consumption – The amount of time needed to execute the computational portions of the application's code, or CPU bursts, on a single node is proportional to the number of nodes of the simulated platform. A popular approach, used for instance in [73, 34], is to replace CPU bursts by simulated delays whenever possible. The hope is

that the simulation time becomes orders of magnitude shorter than the real execution. In this view, SMPI executes the first n occurrences of each burst, and then uses the average burst execution time computed over these n samples as the delay in the simulation for all future occurrences, skipping the corresponding code in the application's execution. Alternatively, rather than computing n samples, the user can specify a confidence interval length and the number of samples is determined so that 95% confidence is achieved. While the current implementation uses the sample average as the simulated delay, it would be straightforward to draw this delay from an empirical distribution determined based on the obtained samples. Regardless of the sampling methodology in use, the user can specify a factor by which delays are scaled to simulate compute nodes faster/slower than the host. Unfortunately, the time to execute each CPU burst n times is proportional to the number of simulated nodes, since each process executes each CPU burst n times. The simulation time is thus linear in the number of simulated nodes, which could make it prohibitively long. Many parallel applications, however, consist of tasks that execute similar CPU bursts (e.g., applications that follow the Single Program Multiple Data paradigm). Therefore, SMPI allows for the measurement of the execution times of the first n occurrence of each CPU burst, regardless of the MPI process. Simulation time is then independent of the number of simulated nodes. Note that for applications with data-dependent performance, all CPU bursts may need to be executed. This precludes simulation on a single host and, instead, the user must run the simulation on a cluster using one the SPDAs mentioned in Section 2.3. For non-data-dependent applications, experiments in [22] conducted for the EP NAS Parallel Benchmark show that the above technique can reduce simulation time dramatically while having negligible impact on simulation accuracy.

Reducing RAM consumption – In general, the host cannot accommodate the memory footprint of the simulated application. In an SMPI simulation all MPI processes run as threads that share the same address space. This is enabled by an automatic source-to-source translation performed by a preprocessor integrated with SMPI. Having all processes run as threads (or as ucontexts) allows SMPI to use a technique initially proposed in [1] for removing large array references. Essentially, references to private arrays are replaced by references to a single shared array. If the MPI application has m processes that each uses an array of size s, then the RAM requirement is reduced from $m \times s$ to s. This leads to incorrect application execution, and likely to many race conditions, but the hope is that the performance behavior of the original application is preserved. Because it corrupts application data, this technique cannot lead to accurate simulation results if the application exhibits data-dependent performance. As for the CPU consumption reduction technique described earlier, for such applications the user must resort to running to simulation on a cluster that can provide the necessary memory capacity. Experiments in [22] show that, for a simulation of various classes of the DT NAS Parallel Benchmark, memory footprint of the simulation is reduced on average by more than a factor 10 and up to a factor 40, without significant accuracy losses.

6 Simulation Usability

An impediment to obtaining scientifically sound results with SPDAs is the lack of available, open and maintained simulation framework. Our goal is for SIMGrid to be recognized as such a framework. Another impediment is the lack of transparent simulation methodologies in the literature, making it impossible to reproduce simulation results obtained by others. Figure 6 depicts the components of a simulation execution.

The central components are the simulated application and the simulation framework, with which the application interacts via one of several APIs in the case of SIMGrid. The application is instantiated based on application-specific configuration parameters. Its execution is simulated based on a description of the simulated platform with compute nodes, network links and topology, as well as optional time-varying resource availabilities and time-varying performance levels delivered by these resources due to out-of-band workloads. The simulation framework produces output in the shape of time-stamped event logs and statistics. Given the size of these logs and the information lost when compiling statistics, visualization capabilities are a useful feature to explore and gain deeper understanding of simulation results, but also to debug the simulation. In this article so far we have focused on the central components in Figure 6. But a simulation framework should be a whole “ecosystem" of tools that ease the entire simulation process, from instantiation to visualization. The SIMGrid project has made several advances in this context, three of which are highlighted in the next sections.

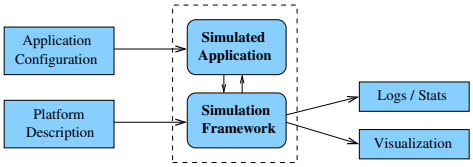


Figure 6: Components of a classical simulation environment.

6.1 Platform Description Instantiation

When using a SPDA, a question faced by all users is that of which platform to simulate, i.e., what platform configuration description should be constructed and provided to the SPDA. The content of this description varies depending on the target domain. For instance, researchers in the peer-to-peer community focus on network topology and network distance between peers, while often ignoring hardware characteristics of the peers themselves save for their availability. By contrast, volunteer computing simulations often ignore network topology issues and solely focus on end-point characteristics, such as compute speed, availability, and network card bandwidth. Platform descriptions used in simulations for HPC computing on cluster and grid platforms strike somewhat of a compromise, with information on both the network and the end-points, even though the network topology is either inherently simpler or intentionally simplified when compared to peer-to-peer simulations.

One option is to instantiate a simulated platform description that reflects a particular real-world platform. Even assuming that it is possible to discover full configuration information, only a limited number of real-world platforms are available. And yet, in most cases, users wish to run large numbers of simulations for large numbers of platform configurations to reduce the risk of idiosyncrasies biasing the study. Furthermore, it is typically interesting to study how simulation results vary when fundamental platform characteristics evolve. While simulating various subsets of a real-world platform can be done, a preferred approach is to synthesize platform descriptions with characteristics that are representative of (classes of) real-world platform configurations. Consequently, platform synthesis tools have been developed in various communities. For instance, several generators of synthetic internet topologies are available, such as Tiers [16] or BRITE [48]. In the area of cluster computing, the work in [40] proposes a synthetic

method to instantiate representative cluster configurations (number of compute nodes, number of cores per node, memory size and cache size per node, clock frequency, etc.) based on a pool of real-world production clusters. No single synthesizer meets user needs across multiple research communities, but they each provide important pieces that can be combined together. The GridG synthesizer [45] is one attempt toward such a combined solution for synthesizing grid platforms.

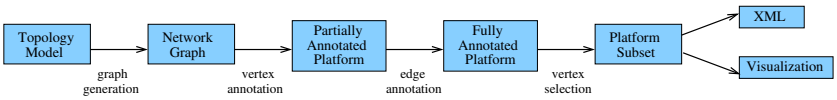


Figure 7: Platform synthesis with SIMULACRUM.

As part of the SIMGrid project we have developed a platform synthesizer called SIMULACRUM [57], which combines the above approaches and attempts to provide a solution applicable across various domains. This solution consists of multiple steps, as depicted in Figure 7, and the user can be involved at each step to provide domain-specific directives. The user goes through all these steps while interacting with the SIMULACRUM graphical user interface. The first step consists in generating a network graph. SIMULACRUM allows the user to choose among multiple topology models. Beyond classical topologies (star, clique, etc.), it implements those popular models that spread vertices over a unit square and connect two vertices u and v via an edge with probability $P(u, v)$. This probability can be based on the uniform or exponential distributions, or on well-known distributions used in [70] and [71]. SIMULACRUM also implements models based on node degree, such as the one proposed in [5], that match the power laws observed in real internet topologies. In its current version, SIMULACRUM does not implement explicitly hierarchical models such as that used in the Tiers [16] synthesizer.

Once an abstract network graph has been constructed, vertices are annotated with qualitative information stating for each vertex whether it is a router, a host, or a cluster. Initially all vertices are considered to be routers. Consider a user who desires a scenario in which half of the one-degree vertices are personal computers, the other half of these vertices are homogeneous compute clusters, vertices with degree between 2 and 4 are powerful servers, and vertices with degree higher than 4 are network routers. The difficulty consists in providing abstractions and mechanisms by which users can easily guide the synthesis to achieve such objectives. To this end SIMULACRUM uses a *promoter* abstraction. A promoter is essentially a guarded decision rule that applies a qualitative annotation to a vertex that is currently not annotated. A set of default promoters is provided (e.g., host, server, homogeneous cluster), but user-defined annotations are also supported. In fact, the user can easily define annotations that assign arbitrary (key,value) pairs to a vertex. The guard, or *filter*, is a conditional expression that specifies under which conditions the rule should be applied. Filters are written as Boolean expressions involving vertex properties (e.g., degree) and platform property (e.g., annotation counts). By defining a sequence of promoters, the user can implements complex scenarios with little effort. For instance, the above example would be obtained with the following promoters:

Promoter 0: AND(node is leaf, probability 0.5) $\Rightarrow$ personal computer
Promoter 1: node is leaf $\Rightarrow$ homogeneous cluster
Promoter 2: degree$\in [2, 4] \Rightarrow$ server

Finally, for each annotation, a resource synthesizer can be used to generate resource characteristics (e.g., fixed values, random values sampled from simple distributions, real-world values picked from a set of production resources, values obtained using other synthesizers such as [40]). The third step of the platform synthesis, edge annotation, uses the same promoter mechanism to assign latency and bandwidth values to each network edge.

After the first three steps, SIMULACRUM has generated a full platform description that can be used to drive a simulation. At this point, the platform may still not possessed all the properties desired by the user. For this reason, SIMULACRUM makes it possible to extract simulation scenarios, each defined by a subset of the full platform. This filtering of vertices and edges can be done manually by the user. After each such modification, the user is notified about the impact on overall platform statistics. Because manuals modification are labor-intensive, SIMULACRUM also provides automated subset selection based on user-provided users. Many default filters are available, e.g., the total number of vertices, the network diameter, the fraction of vertices with a particular annotation. Several filters are provided that pertain to the compute speed of those vertices that are annotated as compute resources. These filters can be used to ensure that the subset contain only resources with compute speeds in a certain range, with a given first (mean), second (variance), third (skewness), and/or fourth (kurtosis) statistical moment of these compute speeds. Users can define their own subset filters by providing short Java classes (typically about 25 lines of code without counting the actual filtering code). SIMULACRUM implements several techniques to quickly identify matching clusters among the 2^n possible subsets in an n-vertex platform. As a last step, the user can export the generated platform to an XML representation. This representation can be passed as is to any simulation built with the SIMGrid framework. It can also be visualized using the standard graph drawing tool `dot`.

By using custom combinations of filters, users can use SIMULACRUM to conveniently synthesize platform descriptions that are relevant across all domains of application of SIMGrid.

6.2 Simulation Visualization

Exploring simulation results in depth, whether to better understand the behavior of the simulated application or to identify bugs in its implementation, is often only tractable using visualization capabilities. Visualization of application events throughout time, based on a trace collected during application execution, is commonplace in the parallel computing community. An advantage of executing the application in simulation is that execution traces can contain precise information on all simulated resources, without changing the natural behavior of the application or introducing probe effects and bugs (so-called Heisenbugs). These traces thus lend themselves to visualization that goes beyond what can be achieved with traces obtained from real-world application execution. The challenge is for visualization to be both informative and scalable. The standard visual representation of application execution is the zoomable timeline view termed "Gantt chart," for instance as seen in traditional performance analysis tools for visualizing the execution of message passing applications [50, 8]. While it is straightforward to use such tools for visualizing traces generated by a SIMGrid simulation, in what follows we describe two more scalable and informative visual representations. These techniques are implemented as part of the Triva [63] toolkit that is developed in part to provide visualization capabilities to SIMGrid simulators.

6.2.1 Squarified Treemaps

A squarified treemap [12] is a space-filling diagram used to represent a tree whose vertices are annotated by numerical values. The visual representation is built by dividing screen space among the children of a vertex recursively, starting with the root vertex. Each vertex is associated a rectangle region with an area proportional to the vertex annotation. This region can be subdivided into multiple rectangular sub-regions, based on secondary vertex annotations. The user can navigate the hierarchy to explore various levels of details (e.g., viewing a whole multi-cluster platform as a single region, viewing each cluster as a region whose size is proportional to the cluster's compute power, viewing each compute node as a region proportional to the node's compute power).

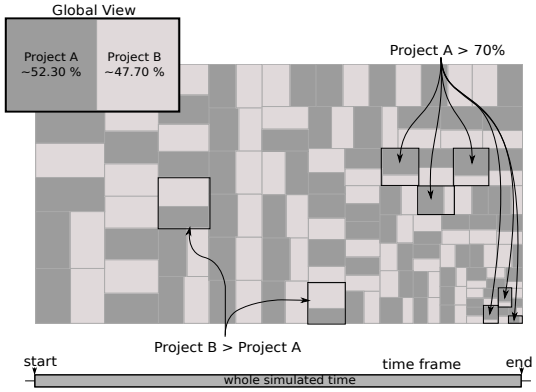


Figure 8: Squarified treemap representation example.

To illustrate the use of squarified treemaps, we present a case study for a volunteer computing simulation similar to that described in Section 5.1. A team of researchers was in the process of developing a simulation of a BOINC [2] system. The goal of this simulation was to explore new work sharing policies and study their impact on overall system throughput and responsiveness. In the early development stages of the simulator, squarified treemaps were used to explore how simulated hosts performed work units. Such a treemap is shown in Figure 8, for a small-scale experiment with 65 hosts and two competing projects named Project A and Project B. Each region of the treemap corresponds to a simulated host, and its area is proportional to the host's delivered compute power. Each host region is subdivided into two sub-regions, one for each project. The size of each project sub-region is proportional to the fraction of work units performed by the host for the corresponding project (dark-colored for Project A and light-colored for Project B). In the top-left corner of the figure is shown a top-level view with a single region for the whole platform. The simulated work sharing policy in this experiment is fair sharing. The global view shows that 52.3% of the work units performed by the platform are for project Project A. While this may seem like a small departure from a fair share policy, examining the treemap representation shows that several "small" hosts, i.e., those with low delivered compute power, perform significantly more Project A work units than Project B work units (more than 70%). The origin of the problem was then quickly identified as a bug in the algorithm for measuring the project-specific compute time for the simulated hosts. Identifying a correlation between low host availability and unfair work unit execution without the

treemap visualization would have been at best difficult, and yet it was immediate based on Figure 8.

6.2.2 Topology-based Visualization

Triva implements a novel visualization capability, called topology-based visualization, that can help the user pin-point resource contention and understand its effect on application execution. In this visualization, depicted in Figure 9, hosts are represented by squares and network links by diamonds. Sizes are proportional to CPU power or network bandwidth, and resource utilization is shown with a gray fill. Utilization is computed as an average over a configurable time slice, which is depicted as a fraction of the overall execution time in the top-left corner of the view.

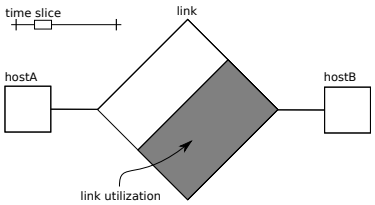


Figure 9: Topological representation example.

To demonstrate how such visualization can be used to reveal performance issues for simulated application, we present an example for an on-line simulation of the DT NAS Parallel benchmark (class A, using the White Hole algorithm) [4]. The execution is simulated on two clusters connected via a few routers and network links, as depicted in Figure 10. The main view corresponds to utilization computed as an average over the entire application execution, while the three small views at the bottom of the figure are for smaller time slices at different stages of the execution. With such visualization, is is straightforward to see that the links interconnecting clusters are the performance bottleneck and that they are saturated during the whole execution. It turns out that, in this execution, application processes are mapped to compute nodes taken in an arbitrary order (i.e., the order in which they are listed in a configuration file). Instead, processes should be mapped to compute nodes judiciously so as to promote communication locality. For the DT NAS benchmark with the White Hole algorithm, communication locality is easily achieved by placing so-called forwarder processes close to data sources, thereby shortening communication paths and avoiding communication between the two clusters unless absolutely necessary. Simulating the application again with a thus modified list of compute nodes leads to a view in which the inter-cluster network links have low contention, i.e., mostly white, while intra-cluster links carry most of the communication workload. Although this is a known example of a communication bottleneck, we contend that such visualization can be used effectively for performance debugging of applications with complex communication patterns running on complex platform topologies. Particularly useful is the feature that allows to tune the width of the time slice and to slide it along the time axis.

6.3 Formal Verification

One advantage of simulation is that it allows for deterministic (and reproducible) execution paths of the simulated application. Consequently, debugging the application

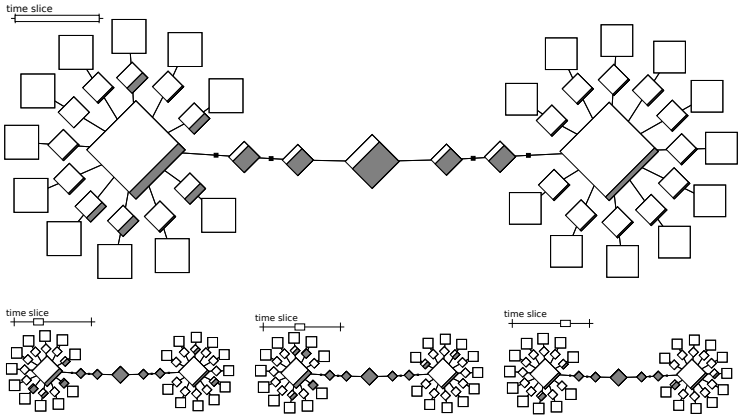


Figure 10: Topology-based views for a simulated execution of the DT NAS Parallel Benchmark (class A, White Hole) on two clusters, for an arbitrary mapping of processes to compute nodes.

is much easier in simulation than via real-world experiments that often cause non-deterministic executions. Nevertheless, simulation solves only part of the distributed application debugging challenge: the developer still has to determine which set of simulation experiments is sufficient to cover all relevant situations that may expose bugs. Because simulation is not exhaustive with respect to all possible interleavings of communication events, a program that behaves correctly in simulation can still fail when deployed in real life due to execution paths that were "missed" in simulation.

To address this problem, SIMGrid allows for formal verification through *model checking*, which can be used with any simulation written in C with the SIMGrid APIs, *without any source code modifications* [49]. Conceptually, a basic model checker for safety properties (i.e., local assertions) explores the state space of the model looking for invalid states that do not meet the specification. The search continues until a state is found that violates some correctness property, the whole state space is explored, or the model checker runs out of resources. If an invalid state is found, the model checker returns a counter-example in the form of an execution trace. Model checking is often more effective at discovering bugs than traditional testing due to its exhaustive nature, meaning that it considers "corner cases" that might otherwise be overlooked. From a simulation point of view, model checking is equivalent to the simulation of the target application on all possible platform configurations.

In SIMGrid, if model checking is enabled, the main simulation loop is replaced with a state space exploration algorithm that systematically executes all *relevant* interleavings of the communication actions of the simulated processes. The simulated application is annotated with assertions that are evaluated for every state. In the case an assertion is violated, the exploration is terminated and the user is provided with the trace of communication events that produced the violation. Because the number of possible interleavings grows exponentially with the number of processes, SIMGrid implements a state space reduction technique based on *Dynamic partial-order reduction*. Essentially, this technique exploits existing symmetries in the state space of the application, based on the semantics of the communication operations [62]. Thanks to this feature, we were able to identify (and fix) bugs in non trivial simulated programs such as an implementation of Chord [65]. These bugs appeared only sporadically in large-scale simulations, and

visual inspection of the complex execution traces for these simulations did not make it possible to identify the root causes of the problem.

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 improve the accuracy, the scalability, and/or the usability of simulations, with the overall goal of advancing the state of the art of parallel and distributed application simulators. One of the salient aspects of SIMGrid is that it targets multiple domains, including large-scale simulations of peer-to-peer and volunteer computing systems, simulation of abstract parallel and distributed applications, and simulation of legacy applications on clusters. We have described how the aforementioned advances contribute to providing multiple accuracy/scalability trade-offs, which compare favorably with and often are orders of magnitude better than trade-offs achieved by extent domain-specific simulators used by researchers in the field. The SIMGrid user can then pick an appropriate trade-off for a particular simulation domain. More specifically, the user can experimentally maximize simulation accuracy while remaining within scalability constraints. She can then empirically quantify the corresponding loss of accuracy by conducting smaller-scale experiments using the most accurate, but least scalable, simulation models.

The SIMGrid development team will tackle several research directions in the upcoming years. An important one consists in developing new scalable and accurate simulation models including 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 peta/exa-scale HPC platforms. A second direction is to enhance the ecosystem of tools surrounding SIMGrid. Consider for instance our simulation visualization framework. Although simulation allows for deterministic comparison of competing algorithms, the complexity and scale of the simulation often makes it difficult to truly understand why one algorithm is better than the other. Providing not only execution trace visualization but also scalable trace *comparison* visualization will prove invaluable for drawing conclusions from simulation results. Finally, a third 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 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.

To date, most simulation results in the parallel and distributed computing literature are obtained with simulators that are ad-hoc, unavailable, and/or no longer maintained. Furthermore, simulation methodology is rarely transparent or well documented. There is thus a strong need for recognized simulation frameworks that foster "Open Science"

by which simulation results can be reproduced and further analyzed. Our goal if for SIMGrid to fill this need, relying both on the accomplishment described in this article and on the future accomplishments highlighted above. The SIMGrid software welcomes contributors and is publicly available at `http://simgrid.gforge.inria.fr`.

8 Acknowledgments

This work has been supported by *ANR* (French National Agency for Research) through project references ANR 08 SEGI 022 (USS SimGrid) and ANR 07 JCJC 0049 (DOCCA), 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. The authors would like to thank *CNPq* (Brazilian National Counsel of Technological and Scientific Development) for funding the PhD thesis of Pedro Velho. The authors would like to thank the Grid5000 [11] project that has provided platforms for conducting experiments.

References

[1] V. S. Adve, R. Bagrodia, E. Deelman, and R. Sakellariou. Compiler-Optimized Simulation of Large-Scale Applications on High Performance Architectures. *Journal of Parallel and Distributed Computing*, 62(3):393–426, 2002.

[2] D. P. Anderson. BOINC: a System for Public-Resource Computing and Storage. In *Proc. of the 5th IEEE/ACM Intl. Workshop on Grid Computing*, pages 4–10, 2004.

[3] Rajive Bagrodia, Ewa Deelman, and Thomas Phan. Parallel Simulation of Large-Scale Parallel Applications. *Intl. Journal of High Performance Computing Applications*, 15(1):3–12, 2001.

[4] D.H. Bailey, E. Barszcz, J.T. Barton, D.S. Browning, R. L. Carter, L. Dagum, R.A. Fatoohi, P.O. Frederickson, T.A. Lasinski, R.S. Schreiber, H.D. Simon, V. Venkatakrishnan, and S.K. Weeratunga. The Nas Parallel Benchmarks. *Intl. Journal of High Performance Computing Applications*, 5(3):63–73, Sep 1991.

[5] Albert-Lázló Barabási and Réka Albert. Emergence of Scaling in Random Networks. *Science*, 286:509–512, October 1999.

[6] Ingmar Baumgart, Bernhard Heep, and Stephan Krause. OverSim: A Flexible Overlay Network Simulation Framework. In *Proc. of the 10th IEEE Global Internet Symp. (GI)*, pages 79–84. IEEE, May 2007.

[7] Olivier Beaumont, Lionel Eyraud-Dubois, and Young Joon Won. Using the Last-mile Model as a Distributed Scheme for Available Bandwidth Prediction. In *Proc. of the 17th Intl. European Conf. on Parallel and Distributed Computing (EuroPar)*, volume 6852 of *Lecture Notes in Computer Science*, pages 103–116. Springer-Verlag, August 2011.

[8] R. Bell, A.D. Malony, and S. Shende. ParaProf: A Portable, Extensible, and Scalable Tool for Parallel Performance Profile Analysis. In *Proc. of the 9th Intl. Euro-Par Conf. on Parallel Processing*, volume 2790 of *Lecture Notes in Computer Science*, pages 17–26. Springer, 2003.

[9] William H. Bell, David G. Cameron, Luigi Capozza, A. Paul Millar, Kurt Stockinger, and Floriano Zini. OptorSim - A Grid Simulator for Studying Dynamic Data Replication Strategies. *Intl. Journal of High Performance Computing Applications*, 17(4):404–416, 2003.

[10] Dimitri P. Bertsekas and Robert G. Gallager. *Data Networks*. Prenctice Hall, second edition, 1996.

[11] Raphaël Bolze, Franck Cappello, Eddy Caron, Michel Daydé, Frédéric Desprez, Emmanuel Jeannot, Yvon Jégou, Stephane Lanteri, Julien Leduc, Noredine Melab, Guillaume Mornet, Raymond Namyst, Pascale Primet, Benjamin Quetier, Olivier Richard, El-Ghazali Talbi, and Iréa Touche. Grid'5000: A Large Scale And Highly Reconfigurable Experimental Grid Testbed. *Intl. Journal of High Performance Computing Applications*, 20(4):481–494, 2006.

[12] Mark Bruls, Kees Huizing, and Jarke van Wijk. Squarified Treemaps. In *Proc. of the Joint Eurographics and IEEE TCVG Symp. on Visualization*, pages 32–42, 2000.

[13] John S. Bucy, Jiri Schindler, Steven W. Schlosser, Gregory R. Ganger, and Contributors. The DiskSim Simulation Environment Version 4.0 Reference Manual. Technical Report CMU-PDL-08-101, Carnegie Mellon University, Parallel Data Lab, 2008.

[14] Rajkumar Buyya and Manzur 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):1175–1220, December 2002.

[15] Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, Cesar A. F. De Rose, and Rajkumar Buyya. CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms. *Software: Practice and Experience*, 41(1):23–50, January 2011.

[16] Kenneth Calvert, Matthew Doar, and Ellen Zegura. Modeling Internet Topology. *IEEE Communications Magazine*, 35(6):160–168, June 1997.

[17] Henri Casanova. Simgrid: A Toolkit for the Simulation of Application Scheduling. In *Proc. of the first IEEE Intl. Symp. on Cluster Computing and the Grid (CCGrid)*, pages 430–437. IEEE Computer Society, May 2001.

[18] Henri Casanova, Arnaud Legrand, and Loris Marchal. Scheduling Distributed Applications: the SimGrid Simulation Framework. In *Proc. of the third IEEE Intl. Symp. on Cluster Computing and the Grid (CCGrid)*, pages 138–145. IEEE Computer Society, May 2003.

[19] Henri Casanova and Loris Marchal. A Network Model for Simulation of Grid Application. Technical Report 2002-40, École Normale Supérieure de Lyon, LIP, 2002.

[20] Dah-Ming Chiu. Some Observations on Fairness of Bandwidth Sharing. In *Proceeding of the 5th IEEE Symp. on Computers and Communications (ISCC)*, pages 125–131, Antibes, France, July 2000. IEEE Computer Society.

[21] Brent Chun, David Culler, Timothy Roscoe, Andy Bavier, Larry Peterson, Mike Wawrzoniak, and Mic Bowman. PlanetLab: an Overlay Testbed for Broad-Coverage Services. *ACM SIGCOMM Computer Communication Review*, 33(3):3–12, 2003.

[22] Pierre-Nicolas Clauss, Mark Stillwell, Stéphane Genaud, Frédéric Suter, Henri Casanova, and Martin Quinson. Single Node On-Line Simulation of MPI Applications with SMPI. In *Proc. of the 25th IEEE Intl. Parallel and Distributed Processing Symp (IPDPS)*, pages 661–672. IEEE, May 2011.

[23] Phillip Dickens, Philip Heidelberger, and David Nicol. Parallelized Direct Execution Simulation of Message-Passing Parallel Programs. *IEEE Transactions on Parallel and Distributed Systems*, 7(10):1090–1105, 1996.

[24] Marcel Dischinger, Andreas Haeberlen, P. Krishna Gummadi, and Stefan Saroiu. Characterizing Residential Broadband Networks. In *Proc. of the 7th ACM SIGCOMM Conf. on Internet Measurement*, pages 43–56. ACM, October 2007.

[25] C. Dobre, F. Pop, and V. Cristea. New Trends in Large Scale Distributed Systems Simulation. *Journal of Algorithms & Computational Technology*, 5(2):221–257, 2011.

[26] Bruno Donassolo, Henri Casanova, Arnaud Legrand, and Pedro 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)*, pages 605–612, June 2010.

[27] Trilce Estrada, Michela Taufer, Kevin Reed, and David P. Anderson. EmBOINC: An Emulator for Performance Analysis of BOINC Projects. In *Proc. of the 3rd Workshop on Desktop Grids and Volunteer Computing Systems (PCGrid)*, 2009.

[28] Kayo Fujiwara and Henri Casanova. Speed and Accuracy of Network Simulation in the SIMGrid Framework. In *Proc. of the First Intl. Workshop on Network Simulation Tools (NSTools)*. ACM, 2007.

[29] 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, and 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*, volume 3241 of *Lecture Notes in Computer Science*, pages 97–104. Springer, September 2004.

[30] Pedro García, Carles Pairot, Rubén Mondéjar, Jordi Pujol, Helio Tejedor, and Robert Rallo. PlanetSim: A New Overlay Network Simulation Framework. In *Proc. of the 4th Intl. Workshop on Software Engineering and Middleware (SEM)*, volume 3437 of *Lecture Notes in Computer Science*, pages 123–137, September 2004.

[31] Thomer M. Gil, Frans Kaashoek, Jinyang Li, Robert Morris, and Jeremy Stribling. P2PSim, a Simulator for Peer-to-Peer Protocols. Available at `http://pdos.csail.mit.edu/p2psim/`, 2005.

[32] William 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*, volume 2474 of *Lecture Notes in Computer Science*. Springer, October 2002.

[33] William Gropp, Ewing Lusk, and Anthony Skjellum. *Using MPI: Portable Parallel Programming with the Message Passing Interface*. Scientific And Engineering Computation Series. MIT Press, 2nd edition, 1999.

[34] Duncan Grove and Paul Coddington. Communication Benchmarking and Performance Modelling of MPI Programs on Cluster Computers. *The Journal of Supercomputing*, 34(2):201–217, 2005.

[35] Erik Heien, Noriyuki Fujimoto, and Kenishi Hagihara. Computing Low Latency Batches with Unreliable Workers in Volunteer Computing Environments. In *Proc. of the Second Workshop on Large-Scale, Volatile Desktop Grids*, April 2008.

[36] Marc-André Hermanns, Markus Geimer, Felix Wolf, and Brian Wylie. Verifying Causality between Distant Performance Phenomena in Large-Scale MPI Applications. In *Proc. of the 17th Euromicro Intl. Conf. on Parallel, Distributed and Network-based Processing (Euro-PDP)*, pages 78–84, February 2009.

[37] Torsten Hoefler, Christian Siebert, and Andrew Lumsdaine. LogGOPSim - Simulating Large-Scale Applications in the LogGOPS Model. In *Proc. of the ACM Workshop on Large-Scale System and Application Performance*, pages 597–604, June 2010.

[38] ningNing. Hu, Li. Li, Zhuoquing Mao, Peter Steenkiste, and Jia Wang. A Measurement Study of Internet Bottlenecks. In *Proc. of the 24th Annual Joint Conf. of the IEEE Computer and Communications Societies (INFOCOM)*. IEEE, March 2005.

[39] Márk Jelasity, Alberto Montresor, Gian Paolo Jesi, and Spyros Voulgaris. PeerSim. Available at `http://peersim.sourceforge.net/`.

[40] Yang-Suk Kee, Henri Casanova, and Andrew Chien. Realistic Modeling and Synthesis of Resources for Computational Grids. In *Proc. of ACM/IEEE SuperComputing 2004 (SC'04)*, November 2004.

[41] Derrick Kondo. SimBOINC: A Simulator for Desktop Grids and Volunteer Computing Systems. Available at `http://simboinc.gforge.inria.fr/`, 2007.

[42] Derrick Kondo, Bahman Javadi, Alexandru Iosup, and Dick Epema. The Failure Trace Archive: Enabling Comparative Analysis of Failures in Diverse Distributed Systems. In *Proceeding of the 10th IEEE/ACM Intl. Symp. on Cluster, Cloud and Grid Computing (CCGrid)*, pages 398–407. IEEE, May 2010.

[43] Dawn Leaf. NIST Cloud Computing Program Overview. Available at `http://www.nist.gov/itl/cloud/upload/Leaf-CCW-II-2.pdf`, November 2010.

[44] Edgar León, Rolf Riesen, and Arthur Maccabe. Instruction-Level Simulation of a Cluster at Scale. In *Proc. of the Intl. Conf. for High Performance Computing and Communications (SC)*, November 2009.

[45] Dong Lu and Peter Dinda. Synthesizing Realistic Computational Grids. In *Proc. of the ACM/IEEE SC2003 Conf. on High Performance Networking and Computing (SC)*, November 2003.

[46] Laurent Massoulié and James Roberts. Bandwidth Sharing: Objectives and Algorithms. In *Proc. of the Eighteenth Annual Joint Conf. of the IEEE Computer and Communications Societies on Computer Communications (INFOCOM)*, volume 3, pages 1395–1403, March 1999.

[47] S. Mccanne, S. Floyd, and K. Fall. The Network Simulator (s2). Available at `http://nsnam.isi.edu/nsnam`.

[48] Alberto Medina, Anukool Lakhina, Ibrahim Matta, and John Byers. BRITE: An Approach to Universal Topology Generation. In *Proc. of the Intl. Workshop on Modeling, Analysis and Simulation of Computer and Telecommunications Systems (MASCOTS)*, August 2001.

[49] Stephan Merz, Martin Quinson, and Cristian Rosa. Simgrid MC: Verification Support for a Multi-API Simulation Platform. In *Proc. of the 31th Formal Techniques for Networked and Distributed Systems – FORTE 2011*, pages 274–288, June 2011.

[50] M.S. Muller, A. Knupfer, M. Jurenz, M. Lieber, H. Brunst, H. Mix, and W. E. Nagel. Developing Scalable Applications with Vampir, VampirServer and VampirTrace. *Parallel Computing: Architectures, Algorithms and Applications*, 38:637–644, 2007.

[51] Stephen Naicken, Anirban Basu, Barnaby Livingston, and Sethalat Rodhetbhai. Towards Yet Another Peer-to-Peer Simulator. In *Proc. of the Fourth Intl. Working Conf. Performance Modelling and Evaluation of Heterogeneous Networks (HET-NETs)*, September 2006.

[52] The ns-3 Network Simulator. `http://www.nsnam.org`.

[53] Alberto Núñez, Javier Fernández, José Daniel García, Félix García, and Jesús Carretero. New Techniques for Simulating High Performance MPI Applications on Large Storage Networks. *Journal of Supercomputing*, 51(1):40–57, 2010.

[54] Simon Ostermann, Radu Prodan, and Thomas Fahringer. Dynamic Cloud Provisioning for Scientific Grid Workflows. In *Proc. of the 11th ACM/IEEE Intl. Conf. on Grid Computing (Grid)*, October 2010.

[55] Brad Penoff, Alan Wagner, Michael Tüxen, and Irene Rüngeler. MPI-NetSim: A Network Simulation Module for MPI. In *Proc. of the 15th Intl. Conf. on Parallel and Distributed Systems (ICPADS)*, pages 464–471, December 2009.

[56] Martin Quinson. GRAS: a Research and Development Framework for Grid and P2P Infrastructures. In *Proc. of the IASTED Intl. Conf. on Parallel and Distributed Computing and Systems*. Acta Press, 2006.

[57] Martin Quinson, Laurent Bobelin, and Frédéric Suter. Synthesizing Generic Experimental Environments for Simulation. In *Proc. of the 5th Intl. Conf. on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC)*, pages 222–229, November 2010.

[58] Kavitha Ranganathan and Ian Foster. Decoupling Computation and Data Scheduling in Distributed Data-Intensive Applications. In *Proc. of the 11th IEEE Intl. Symp. on High Performance Distributed Computing (HPDC)*. IEEE Computer Society, 2002.

[59] Ralf Reussner, Peter Sanders, and Jesper Larsson Träff. SKaMPI: a Comprehensive Benchmark for Public Benchmarking of MPI. *Scientific Programming*, 10(1):55–65, 2002.

[60] Rolf Riesen. A Hybrid MPI Simulator. In *Proc. of the IEEE Intl. Conf. on Cluster Computing*, September 2006.

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

[62] Cristian Rosa, Stephan Merz, and Martin Quinson. A Simple Model of Communication APIs – Application to Dynamic Partial-order Reduction. In *Proc. of the 10th Intl. Workshop on Automated Verification of Critical Systems – AVOCS 2010*, pages 137–151, September 2010.

[63] L. M. Schnorr, G. Huard, and P. O. A. Navaux. Triva: Interactive 3D Visualization for Performance Analysis of Parallel Applications. *Future Generation Computer Systems Journal*, 26(3):348–358, 2010.

[64] Spyros Sioutas, George Papaloukopoulos, Evangelos Sakkopoulos, Kostas Tsichlas, and Yannis Manolopoulos. A Novel Distributed P2P Simulator Architecture: D-P2P-Sim. In *Proc. of the 18th ACM Conf. on Information and Knowledge Management (CIKM)*, pages 2069–2070, November 2009.

[65] Ion Stoica, Robert Morris, David Liben-Nowell, David Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord: a Scalable Peer-to-Peer Lookup Protocol for Internet Applications. *IEEE/ACM Trans. on Networking*, 11(1):17–32, 2003.

[66] Michela Taufer, Andre Kerstens, Trilce Estrada, David Flores, and Patricia J. Teller. SimBA: A Discrete Event Simulator for Performance Prediction of Volunteer Computing Projects. In *Proc. of the 21st Intl. Workshop on Principles of Advanced and Distributed Simulation (PADS)*, pages 189–197. IEEE Computer Society, 2007.

[67] Mustafa Tikir, Michael Laurenzano, Laura Carrington, and Allan Snavely. PSINS: An Open Source Event Tracer and Execution Simulator for MPI Applications. In *Proc. of the 15th Intl. Euro-Par Conf. on Parallel Processing*, number 5704 in Lecture Notes in Computer Science, pages 135–148. Springer, August 2009.

[68] András Varga. The OMNeT++ Discrete Event Simulation System. In *Proc. of the 15th European Simulation Multiconference (ESM)*, June 2001.

[69] Pedro Velho and Arnaud 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)*, March 2009.

[70] Bernard Waxman. Routing of Multipoint Connections. *IEEE Journal on Selected Areas in Communications*, 6(9):1617–1622, December 1988.

[71] Ellen Zegura, Kenneth Calvert, and Michael Donahoo. A Quantitative Comparison of Graph-based Models for Internet Topology. *IEEE/ACM Transactions on Networking*, 5(6):770–783, December 1997.

[72] Jidong Zhai, Wenguang Chen, and Weimin Zheng. PHANTOM: Predicting Performance of Parallel Applications on Large-Scale Parallel Machines Using a Single Node. In *Proc. of the 15th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming (PPOPP)*, pages 305–314, January 2010.

[73] Gengbin Zheng, Gunavardhan Kakulapati, and Laxmikant Kale. BigSim: A Parallel Simulator for Performance Prediction of Extremely Large Parallel Machines. In *Proc. of the 18th Intl. Parallel and Distributed Processing Symp. (IPDPS)*, April 2004.

Centre de recherche INRIA Nancy – Grand Est
LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399