SimGrid: a Generic Framework for Large-Scale Distributed Experiments

Martin Quinson

SimGrid: a Generic Framework for Large-Scale Distributed Experiments

Martin Quinson – Nancy University/Loria, France.
Martin.Quinson@loria.fr

1 Introduction

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

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

The use of real-world platforms proves time- and labor-intensive while not allowing reproducibility. Emulators, although attractive, often prove too cumbersome and are rarely used by researchers in our area. Packet-level simulators from the networking community prove poorly adapted to our purpose. Consequently, the majority of published results are obtained in simulation.

Simulation has been used extensively in several areas of computer science for decades, e.g., for microprocessor design and network protocol design. In these areas, several widely used and acknowledged simulation frameworks are available. By comparison, the use of sound simulation frameworks for distributed applications on distributed computing platforms is not as developed, certainly without any standard simulation tool (although network simulation is a key component of distributed application simulation). An inordinate number of in-house, short-lived and highly specialized simulation tools have been developed to fulfill the need of the community. Expectedly, these tools are difficult to use by others in other contexts than the ones for which they were intended. Moreover, tool proliferation hinders the comparison of (often unreproducible) published results.

In this talk we describe a comprehensive simulation framework, SimGrid, for the simulation of distributed applications on distributed platforms. Our goal is to describe the salient capabilities of SimGrid and explain how they allow users to perform simulations for a wide range of applications and platforms. Some of the key features of SimGrid are: (1) A scalable and extensible simulation engine that implements several validated simulation models, and that makes it possible to simulate arbitrary network topologies, dynamic compute and network resource availabilities, as well as resource failures; (2) High-level user interfaces for distributed computing *researchers* to quickly prototype simulations either in C or in Java; (3) APIs for distributed computing *developers* to develop distributed applications that can seamlessly run in "simulation mode" or in "real-world mode."

This paper is organized as follows. Section 2 provides an overview of SimGrid, while Section 3 provides some insights on quality of SimGrid-based simulations, in term of speed, scalability and accuracy. Finally, Section 4 concludes with perspectives on future work.

2 The SimGrid Framework

The SimGrid project was initiated in 1999 to allow the study of scheduling algorithms for heterogeneous platforms. Ten years after, it has evolved to a generic and modular framework allowing the study of almost any distributed applications.

SimGrid offers four user interfaces which can be sorted in two categories: SimDag and MSG are intended to the researchers wanting to study their algorithms while GRAS and MSG are intended to developpers wanting to improve their applications. The simulation kernel is modular, and several models are pluggable. It supports dynamic resource availabilities and failures.

The **SimDag** API is designed for the investigation of scheduling heuristics for applications as task graphs. It allows to create tasks, add dependencies between tasks, retrieve information about the platform, schedule tasks for execution on resources, and compute the DAG execution time. The **MSG** interface allow the study of CSP applications. It is the most widely used API, perfectly usable for scheduling algorithms, P2P or desktop grids. Java bindings of this interface are provided allowing user reluctant to program in C to still use SimGrid. The **GRAS** system allows the development of distributed applications (e.g., a resource information service) within the simulator, but that are then seamlessly deployable on real-world platforms without code modification. To enable this, the same API is implemented twice (once for the simulated world, once for the real world), and user code simply has to be linked against the appropriate library. **SMPI** enables the direct simulation of unmodified MPI applications by intercepting MPI primitives.

SURF, the simulation engine, was designed with two main goals in mind. First, it must be highly modular to allow the implementation (and comparison) of several resource models. The default one implements a MaxMin sharing strategy for both the network and the CPU. *Vegas* and *Reno* models implement the corresponding TCP algorithms thanks to analytical models. A model for the execution of parallel tasks is also provided. New models can be plugged seamlessly by researchers. In addition, as it constitutes the basis of the whole SimGrid framework, SURF is carefully optimized so as not to hinder simulation speed.

XBT is a generic toolbox. **SimIX** is an internal module factorizing efforts to implement the abstraction of multiple concurrent processes on top of SURF.

3 Experimental Evaluation

Simulator Scalability and Speed The classical metric for a simulator's scalability is the **number of simulated nodes/processes** that are allowed to start. In our case, this is only a matter of available memory since nodes are not mapped into threads by default, but into UNIX98 contexts. This removes any limitation on the amount of threads from the kernel and has allowed us to run simulation with up to 2,000,000 simulated processors on a computer with 16Gb of memory.

We feel however that computation time is more representative of usability in practice. The **network simulation speed** is mesured by the speed to simulate a given amount of flows of a given size. Experiments show that SimGrid clearly outperform the GTNetS packet-level simulator in any scenario. The running time of GTNetS is linear in both the number of flows and their sizes while the running time of SimGrid is only linear in the number of flows. To measure the **overall simulation speed**, we compute the amount of task per second handled by a classical master/slaves scenario on a on a 2Ghz 32-bit Laptop with 1Gb of memory. Experiments show that the simulator delivers more than 10,000 tasks per second. This timings do not depend on the amount of processes in the scenario. The Java bindings deliver about 4,000 tasks per second, which is due to the cost of JNI.

Simulation Accuracy We mesure the simulation accuracy by comparing the timing computed by our simulator to the ones provided by the GTNetS packet-level simulator in several scenarios. For the sake of discussion it is assumed that GTNetS results are 100% correct and any discrepancy between the SimGrid results and the GTNetS results is considered SimGrid "error.". Overall, experiments presented show that: (1) Since SimGrid does not account for TCP's slow-start mechanism, its accuracy is poor for very short messages; (2) When a flow's bandwidth is limited by the flow's RTT, the SimGrid error is very low (below 1%); (3) When a flow's bandwidth is limited by the actual link bandwidth, there seems to be a fixed error of roughly 5%; (4) When network congestion rises, SimGrid leads to optimistic results (i.e., flows receive more bandwidths than they do with GTNetS). Amongst other causes, this comes from the fact that packet dropping by overloaded routers is not modeled in SimGrid. The error seems to be heavy-tailed: even in an highly contented network, we observe that for 50% of the flows the error is below 10%, while for 90% of them, the error remains below 50%. Some outliers however exhibit error up to 100%.

4 Conclusion

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

SimGrid is freely available (LGPL license) from its Web page[1] and comes with several example programs and tutorials. SimGrid uses an extensive regression testing suite ensuring a reasonable level of software quality. It is ported to Linux, Windows, Mac OS X and AIX.

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

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