The SIMGRID Project: Simulation and Deployment of Distributed Applications

Martin Quinson, Henri Casanova, Arnaud Legrand, Kayo Fujiwara

The SIMGRID Project
Simulation and Deployment of Distributed Applications

Arnaud Legrand — CNRS, MESCAL INRIA project, Laboratoire Informatique et Distribution

Martin Quinson — Université Henri Poincaré, Nancy 1, LORIA

Henri Casanova, Kayo Fujiwara — Dept. of Information and Computer Science, University of Hawai'i at Manoa

The development of robust and efficient distributed applications has been a research and engineering challenge for several decades, and has recently reached a new dimension with the advent of large-scale distributed systems. Whether one develops a parallel computational science application with MPI for a commodity cluster, a scientific workflow for high-end grids, a network monitoring application that is supposed to scale to thousands of node in large-scale grids, or a peer-to-peer file-sharing application running on tens of thousands of volatile Internet hosts, one strives to develop distributed algorithms as well as resource allocation strategies that exploit the strengths and tolerate the weaknesses of the underlying computing platform. Unfortunately, doing so entails addressing many challenges. Indeed, reasoning about the performance of an application on a possibly complex computing platform is extremely difficult and analytical models often rely on simplifying and unrealistic assumptions. Consequently, one must resort to direct experimentation. However, performing relevant experiments on real-world platforms can be a difficult proposition. First, a fully implemented application is required, while one would like to test and compare alternate designs and algorithms in the planning stage of application development. Second, experiments are limited to platform configurations at hand, which limits their significance and generality. Third, in many cases experiments are not repeatable (e.g., due to non-deterministic resource sharing with other users/applications), which makes it at best extremely difficult to compare alternate approaches fairly.

In the face of these difficulties, researchers and developers typically use simulation. However, accurate and validated simulation models can be elusive. Furthermore, there is an accepted notion that to be more accurate simulations must be more complex, which makes them must more time consuming (with pure emulation as an extreme). This problem gets exacerbated when studying large-scale, long-running applications on large-scale platforms. Finally, there is the common concern that the code written to simulate the application is markedly different from the real application code, raising two issues: the two codes may in fact behave differently; and the effort spent writing the simulated code is wasted.

This poster presents the SIMGRID project, which seeks to address the aforementioned challenges and issues. The SIMGRID software architecture comprises four main components: SURF, MSG, GRAS, and SMPI. The last three components provide APIs for implementing, simulating and/or deploying distributed applications. The first component, SURF, is a fast and accurate simulation engine. The poster describes all four components in terms of their goals, their usage, and their functionality, including experimental validation results when applicable. We review each components below.

SURF (Simulation Kernel) A key question for the simulation of distributed applications on distributed platforms is that of resource sharing: what fractions of a resource's delivered power (e.g., CPU cycles per second, bytes of data transfered per second) are allocated to application tasks that

utilize this resource concurrently? One possibility for simulating resource sharing is to use low-level simulation models, such as packet-level network simulation (e.g., NS-2) and execution of actual application code on top of a virtualization layer (e.g., Microgrid), so that resource sharing emerges naturally (e.g., via packet interleaving). Unfortunately, this approach is expensive, with simulation times possibly longer than simulated times, which is prohibitive for simulating long-running applications on large-scale platforms. Instead, SURF considers the platform as a set of resources, with each of the simulated concurrent tasks utilizes some subset of these resources. SURF computes the fraction of power that each resource delivers to each task that uses it by solving a constrained maximization problem: allocate as much power as possible to all tasks in a way that *maximizes* the *minimum* power allocation over all tasks, also called *Max-Min* fairness. SURF provides a fast implementation of *Max-Min* fairness. Via *Max-Min* fairness, SURF enables fast and realistic simulation of resource sharing (e.g., TCP flows over multi-link LAN and WAN paths, processes on a CPU). Furthermore, it enables trace-based simulation of dynamic resource availability. Finally, simulated platform configurations are described in XML using a syntax that provides a unified abstract basis for all other components of the SIMGRID project (MSG, GRAS, and SMPI). The poster presents the basics of *Max-Min* fairness, an example simulated platform, and validation results.

MSG (Rapid Distributed Algorithms Prototyping) The MSG module provides an API for the easy prototyping of distributed applications by letting users focus solely on algorithmic issues. Simulations are constructed in terms of concurrent processes, which can be created, suspended, resumed and terminated dynamically, and can synchronize by exchanging data. A major difference between MSG and the next two modules, GRAS and SMPI, is that all simulated processes are in the same address space. This simplifies development of the simulation by allowing convenient communications via global data structures. In other words, while MSG can accurately simulate the interactions taking place in a distributed application, including communication and synchronization delays, the simulated application can be implemented with the convenience of a single address space. The poster motivates MSG and presents an illustrative example of its usage and functionality.

GRAS (Development of Production Distributed Applications) The GRAS module is a complete environment for the rapid development of real-world distributed applications. A major added functionality of GRAS, when compared to MSG, is that applications built with GRAS can run in simulated mode but also in the real world. Furthermore, this does not require any code modification. Key portions of the application code are automatically benchmarked so that the application simulation can be instantiated *on the fly*. As a result, GRAS provides a full-fledge environment to develop and deploy production distributed applications with the comfort of simulation for testing, debugging, and evaluation. GRAS provides a high-performance communication layer that allows easy and efficient cross-architecture communication of complex data structures. GRAS is portable and is available for Linux, Mac OSX, Solaris, AIX, IRIX and was validated on twelve CPU architectures. As for MSG, the poster motivates GRAS and illustrates its use and functionality with an example.

SMPI (MPI Applications and Heterogeneous Platforms) MPI is the standard programming interface for parallel computational science applications. Many researchers and practitioners are interested in studying the behavior of MPI applications in various heterogeneous environments (both networks and CPUs), which they do not have at their disposal. SMPI is meant to enable such studies. Using the same benchmarking techniques as the ones developed in GRAS, users can annotate an existing application to indicate which portion of the application should be simulated, Application execution can then then be simulated on arbitrary heterogeneous platforms (thanks to SURF). Although SMPI is still work in progress, the poster motivates it and presents an example of its future use.

The four components of the SIMGRID framework together provide solutions and tools for many of the challenges faced by researchers and developers when designing, prototyping, evaluating, comparing, and deploying distributed applications, as presented in this poster. A draft version of the poster is included with this submission.