Distributed Application Management using Plush

Jeannie Albrecht, Christopher Tuttle, Alex Snoeren, Amin Vahdat
University of California, San Diego
{jalbrecht, ctuttle, snoeren, vahdat}@cs.ucsd.edu

1 Introduction

Recent computing trends have shown an increase in the demand for large-scale, distributed, federated computing environments. Two of the more popular environments that have emerged are the Grid and PlanetLab. At a high level, these systems are similar in many ways; both are comprised of a set of heterogeneous interconnected machines that allows secure resource sharing for a variety of different users and applications. However, at a lower level, the systems are very distinct in the sense that they were designed to solve different types of problems, and therefore have fundamental differences that make it difficult to develop and deploy applications on both platforms. As a result, application designers and researchers create software that runs on either the Grid or Planetlab, but not both.

We propose to solve this problem by describing a common abstraction for both PlanetLab and Grid applications. Further, we present Plush–a tool that implements the distributed application abstraction by providing a pluggable and extensible infrastructure allowing users to customize their environment for running experiments on both PlanetLab and the Grid.

2 Distributed Application Life Cycle

In order to create a unified environment for running both Grid and PlanetLab services, it is essential to identify the key abstractions that describe the life cycle of any distributed application independent of the platform within which it runs. There are five key components of this life cycle, including *experiment specification*, *resource discovery*, *resource allocation*, *service deployment*, and *application control* (as shown in Figure 1). This section describes the details involved in these components.

2.1 Experiment Specification

The experiment specification identifies all aspects of an application or service execution. For each project, it describes the software components involved in the experiment, including how to access the software, installation methods, and processes to run on each machine. These details are specified using an extensible description language that also captures desired resource specifications, declares how these resources should be selected, and defines any other information needed to correctly instantiate and run the application.

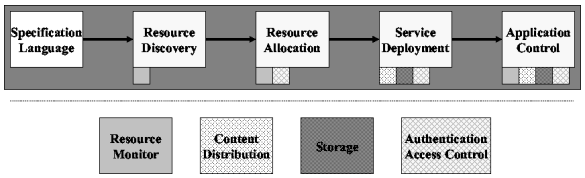

Figure 1: Distributed application lifecycle abstraction.

2.2 Resource Discovery

The goal of the resource discovery phase is to find the best set of physical resources for the distributed application. This stage in the life cycle parses the resource description for each cluster in the experiment specification, and then locates the appropriate set of physical resources to satisfy each request. To accurately fulfill a request for resources, the resource discovery mechanism requires knowledge of the current resource usage for all nodes. From a logical standpoint, the role of the resource monitors is to populate a measurement database that the resource discovery mechanism can later query.

2.3 Resource Allocation

The resource discovery infrastructure interacts directly with the resource allocation system. Typically, the resource discovery system finds a set of resource alternatives, and the resource allocator gets permission to use at least one of those alternatives. The resource allocation methods depends on the administrative domain of the selected hosts. If a system provides a best-effort pool of resources, the resource allocator does nothing. If the system uses a reservation system, the resource allocator acquires a reservation. If the system uses an auction model, the resource allocator submits bids for the desired resources. Regardless of the resource discovery and allocation methods being used, the result is the same. Once this phase has completed, the experiment will have bound to a set of resources that satisfy the specified requirements.

2.4 Service Deployment

The third phase in the distributed application life cycle is service deployment. The purpose of the deployment system is to prepare the physical hosts with the correct

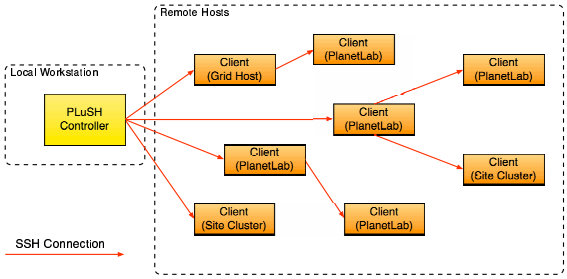

Figure 2: Plush architecture.

processes according to the experiment specification, and then to start the experiment running according to the description. This typically involves copying, uncompressing, and installing the software on the selected hosts. Once the software environments on the hosts are ready, the experiment can be started by running the requested command as defined in the experiment specification.

2.5 Application Control

The fifth and final phase of the application life cycle is the application control phase. This involves monitoring the hosts for failure, monitoring the application for indications of failure, and providing hooks into application-specific code for verifying liveness. The purpose of this component is to maintain application liveness as much as possible, and to provide detailed information about errors, should they arise.

3 Plush Overview

Plush is an extensible execution management system for large-scale distributed systems, including PlanetLab and the Grid. It provides an implementation of the distributed life-cycle abstraction described in the preceding paragraphs. Each phase of the life cycle is viewed as a black box in Plush, where different underlying mechanisms can be swapped out and interchanged without changing the application itself.

3.1 Architecture

The main components of Plush consist of an experiment controller that would typically run on a local workstation, and a light-weight client process that runs on all remote hosts involved in an experiment. The set of clients that a single experiment controller manages is not limited to one platform. As shown in Figure 2, the same controller has the ability to manage clients across all supported platforms, including PlanetLab, the Grid, and any local clusters maintained at the users' site.

When a user starts the Plush experiment controller on a local workstation, they pass in any information needed for authentication and access to resources that allows Plush to create a pool of resources available to the user. The user then passes in an experiment description (XML document) that describes the components needed to run the application. This is the experiment specification phase of the distributed application life cycle.

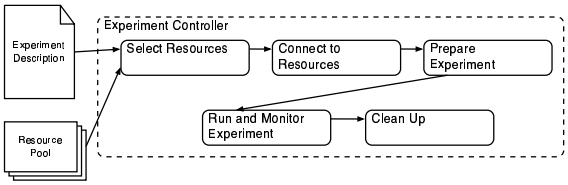

Figure 3: Plush experiment controller.

Next, the controller passes the resource specification from the experiment description on to the resource discovery phase. The resource discovery mechanism finds the resources that meet the specified requirements, and passes the result on to the resource allocator. The allocator determines if the user has access to those resources, and if necessary, makes the appropriate reservations. Once the requested physical resources have been located and reserved, the Plush controller connects to the resources. The controller copies the light-weight client out to all remote hosts creating an underlying communication mesh, and then starts the client process. The controller communicates with the clients to transfer the required software out to all remote hosts to start the service deployment phase.

Application control is the final phase of the life cycle. Here the client processes running on the remote hosts communicate with the main controller to notify the controller of status updates and potential failures. The controller attempts to recover from failures detected. Once the experiment is running, Plush provides monitoring tools to help the user better understand what is happening on the remote hosts. When the experiment has completed, Plush stops all application processes, performs user-specified clean up actions, kills the client process, and disconnects the hosts from the communication mesh. Figure 3 shows this entire process.

3.2 Implementation Status

We have recently released a publicly available version of Plush for application management on PlanetLab. More information on the latest release can be found at `http://ramp.ucsd.edu/projects/plush/`. We are currently working to intergrate application management for the VGrADS execution system (`http://www-csag.ucsd.edu/projects/VGrADS/`) into Plush. We expect to release a version of Plush with these capabilities later this summer.