Application Management and Visualization with Plush

Jeannie Albrecht
Williams College
Williamstown, MA
jeannie@cs.williams.edu

Ryan Braud
UC San Diego
La Jolla, CA
rbraud@cs.ucsd.edu

***Abstract*—Deploying, running, and maintaining applications running on a distributed set of resources is a challenging task. Software developers often spend a significant amount of time dealing with the complexities associated with software configuration and management in these environments. Distributed application management systems are designed to automate the process, and to ultimately help developers cope with the common problems that arise during the design, implementation, and evaluation of distributed systems. In this talk, we highlight the key features of Plush, an application management system for PlanetLab and ModelNet, and describe how Plush simplifies peer-to-peer system visualization and evaluation.**

***Keywords*-Distributed application; PlanetLab; ModelNet; management; visualization**

I. Overview

Most applications deployed on the Internet today run simultaneously on thousands or even millions of computing devices spread around the world. In general, the goal of these *distributed applications* is to connect users to shared resources. One particularly popular type of distributed application is a peer-to-peer (P2P) system. Characterized by functional homogeneity among peers, P2P systems are completely decentralized and, thus, have the ability to scale to large numbers of peers without sacrificing performance. However, while P2P systems offer many benefits, they also introduce new complexities associated with managing computations and services running on hundreds or thousands of computers. For example, consider the task of deploying a P2P application on a large-scale testbed, which involves installing the required software and starting the computation. When running a P2P application on distributed resources, ensuring that hundreds or thousands of computers around the world are all running the correct version of the required software is a cumbersome and tedious task that is further complicated by the heterogeneity—in terms of both hardware and software—of the resources hosting the application.

After installing any required software, additional challenges arise when managing P2P applications. Tasks involved with starting an execution across distributed resources, achieving loose synchronization, keeping an application running, detecting and recovering from failures, and gathering data are all complicated by the distributed nature of a P2P system and the unpredictable behavior of peers. The difficulties associated with these tasks can be frustrating to developers, who end up spending the majority of their time managing executions and trying to detect and react to failures, rather than developing new optimizations and enhancements for increased application performance.

In response to these difficulties, we developed Plush [1], [2] an application management infrastructure that aims to support application management and visualization on different types of resources. Plush simplifies the tasks associated with software configuration, resource management, failure detection, and failure recovery in a variety of distributed execution environments. This is accomplished through an easily-adapted application specification language and resource management system. In this talk, we discuss how Plush works in a general sense, and then describe how Plush specifically supports P2P applications on PlanetLab [3] and ModelNet [4]. We will also demonstrate the functionality of two user interfaces, including a command-line interface and Nebula, a graphical user interface that has been integrated with Plush for application visualization on PlanetLab and ModelNet. The remainder of this paper describes the key subtasks of application management in Plush, including specifying an application, configuring resources and starting an execution, and recovering from failures.

II. Application Specification

The Plush *application specification* is an XML file that describes the flow of control for the application. The XML defines a set of "building block" abstractions that specify the required software packages, processes, and desired resources. The blocks can be arbitrarily combined to support a range of applications in different environments.

Figure 1 shows a sample application specification for a very basic application. Starting at the top, the XML contains information describing the required software, with additional parameters used to specify how to obtain the remote software package. The next section of XML describes the desired resources that define the application's *component*. The third section, called the *application block*, consists of one or more process blocks that specify the execution itself. Any synchronization requirements for the application are described in the application block as well.

```
<?xml version="1.0" encoding="utf-8"?>
<plush>
  <project name="simple">
    <software name="SimpleSoftwareTarball" type="tar">
      <package name="Package" type="web">
        <path>http://sysnet.cs.williams.edu/~jeannie/software.tar</path>
        <dest_path>software.tar</dest_path>
      </package>
    </software>
    <component name="PLMachines">
      <rspec>
        <num_hosts>20</num_hosts>
      </rspec>
      <software name="SimpleSoftwareTarball" />
      <resources>
        <resource type="planetlab" group="williams1"/>
      </resources>
    </component>
    <application name="simple">
      <execution>
        <component_block name="compBlock1">
          <component name="PLMachines" />
          <process_block name="procBlock1">
            <process name="catProc">
              <path>cat</path>
              <cmdline>
                <arg>software.txt</arg>
              </cmdline>
              <cwd/>
            </process>
          </process_block>
        </component_block>
      </execution>
    </application>
  </project>
</plush>
```

Figure 1. Plush application specification that is used to manage an application on 20 PlanetLab machines. This trivial example simply runs "cat software.txt" on each resource.

III. Resource Configuration and Execution

In addition to the application specification, the user must also provide Plush with a resource directory. The resource directory is used to define resource pools in Plush, which are simply groupings of resources that are available to the user and are capable of hosting an application. Plush uses the resources in the resource pool to create a *matching*, or a mapping of resources to an application component. After choosing the resources, Plush can begin connecting to the resources and eventually start the execution.

The Plush architecture consists of a controller and the clients. The Plush controller process is responsible for managing the Plush client processes running on the distributed resources. The controller process is often run on the Desktop computer of the Plush user. The clients are lightweight processes that run on specified ports on each resource involved in an application. When configuring resources and starting an execution, the controller initiates a separate TCP connection to each client process creating a communication fabric. For the remainder of the P2P application's execution, the controller sends messages to the clients via the fabric instructing them to install software, run commands, and start processes on behalf of the user.

IV. Failure Detection and Recovery

After constructing the communication fabric, the client processes running on the distributed resources monitor the liveness of the P2P application. Thus, if any processes exit with an incorrect exit code or issue an application-specific warning, the clients send a message back to the Plush controller. The controller then decides how to recover from the failure, based on the specifications provided by the user. The clients also periodically send the controller updates regarding their individual status and progress. Using these status updates, the Plush controller can construct a single, global view of the progress of a distributed P2P application.

If a client detects a failure and informs the controller, the actions used to recover from the error are chosen based on the severity of the problem. The default options include restarting the failed process, finding a replacement resource, or aborting the either application. Other application-specific failure recovery behaviors are possible via XML-RPC callbacks. In this way, the controller can instruct the client to recover from the failure using application-specific code.

V. Conclusion

In conclusion, Plush is an application management framework that helps developers deploy and maintain software running on distributed set of resources. Through a generic resource management interface and an extensible application specification, Plush supports execution on several different types of resources, including PlanetLab machines and ModelNet emulated clusters. Through two intuitive user interfaces, Plush helps P2P developers deploy, monitor, and visualize the status and progress of applications running on hundreds of distributed resources. Additional information regarding the installation and use of Plush can be found at [5].

References

[1] J. Albrecht, R. Braud, D. Dao, N. Topilski, C. Tuttle, A. C. Snoeren, and A. Vahdat, "Remote Control: Distributed Application Configuration, Management, and Visualization with Plush," in *Proceedings of the USENIX Large Installation System Administration Conference (LISA)*, 2007.

[2] J. Albrecht, C. Tuttle, A. C. Snoeren, and A. Vahdat, "PlanetLab Application Management Using Plush," *ACM Operating Systems Review (OSR)*, vol. 40, no. 1, 2006.

[3] L. L. Peterson, A. C. Bavier, M. E. Fiuczynski, and S. Muir, "Experiences Building PlanetLab." in *Proceedings of the ACM/USENIX Symposium on Operating System Design and Implementation (OSDI)*, 2006.

[4] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostić, J. Chase, and D. Becker, "Scalability and Accuracy in a Large-Scale Network Emulator," in *Proceedings of the ACM/USENIX Symposium on Operating System Design and Implementation (OSDI)*, 2002.

[5] "Plush Webpage," http://plush.cs.williams.edu.