JEANNIE ALBRECHT, RYAN BRAUD, DARREN DAO, NIKOLAY TOPILSKI, CHRISTOPHER TUTTLE, ALEX C. SNOEREN, AND AMIN VAHDAT

managing distributed applications with Plush

BUILDING AND MANAGING DISTRIBUTED applications is difficult. In addition to the usual challenges associated with software development, distributed applications are often designed to run on computers spread across the Internet, and therefore they have to be robust to highly variable32 network conditions and failures that are inevitable in wide-area networked environments. As a result, developers spend a significant portion of their time deploying and debugging distributed applications on remote machines spread around the world. Plush aims to ease this management burden for a broad range of distributed applications in a variety of execution environments.

Jeannie Albrecht is an Assistant Professor of Computer Science at Williams College in Williamstown, Massachusetts. She received her Ph.D. in Computer Science from the University of California, San Diego, in June 2007 under the supervision of Amin Vahdat and Alex C. Snoeren.

jeannie@cs.williams.edu

Ryan Braud is a Ph.D. student at the University of California, San Diego, where he works under the direction of Amin Vahdat in the Systems and Networking Research Group. His interests include high-performance file distribution protocols and tools for building and debugging distributed systems, among others.

rbraud@cs.ucsd.edu

Darren Dao is a graduate student at the University of California, San Diego, where he works under the direction of Amin Vahdat in the Systems and Networking Research Group. He received his B.S. in Computer Science from the University of California, San Diego, in 2006.

hdao@ucsd.edu

Nikolay Topilski is pursuing his M.S. in Computer Science at the University of California, San Diego, where he also received his B.S. He works with Amin Vahdat, and his area of concentration is distributed network applications.

ntopilski@gmail.com

Christopher Tuttle is a Software Engineer at Google in Mountain View, California. He received his M.S. in Computer Science from the University of California, San Diego, in December 2005 under the supervision of Alex C. Snoeren.

ctuttle@google.com

Alex C. Snoeren is an Assistant Professor in the Computer Science and Engineering Department at the University of California, San Diego, where he is a member of the Systems and Networking Research Group. His research interests include operating systems, distributed computing, and mobile and wide-area networking.

snoeren@cs.ucsd.edu

Amin Vahdat is a Professor in the Department of Computer Science and Engineering and the Director of the Center for Networked Systems at the University of California, San Diego. Before joining UCSD in January 2004, he was on the faculty at Duke University from 1999 to 2003.

vahdat@cs.ucsd.edu

No one can deny the success of the Internet. With the growing popularity of pocket-sized network-capable devices such as the iPhone, the Internet has become an integral part of our society, working its way into every aspect of our lives. As the number of Internet users continues to increase, the user demand for Internet-based services, including banking Web sites, news Web sites, and search engines, also increases. In response to this growth, many of these services require more computing power to achieve acceptable levels of performance. In short, companies need more than a single computer—or even a single room full of computers—to satisfy the computing needs of their customers. Thus, more and more services are turning to *distributed applications*, which spread their workload among a distributed set of computers, to help meet the user demand.

Distributed applications have the potential to drastically improve the scalability, fault tolerance, and reliability achieved by an Internet-based service. However, building distributed applications also introduces many new challenges to software developers. When using a distributed set of resources, developers need mechanisms for locating and configuring remote computers and for detecting and recovering from the failures that are inherent in distributed environments. In response to these challenges, many developers write complex scripts to help automate the tasks of connecting to resources, installing the needed software, starting the execution, and monitoring performance. Most of these scripts are customized to work for a specific application in a specific execution environment,

and thus they are not easily extended to help other developers with similar goals.

We want to find a solution to this problem by providing a general-purpose distributed application management system that simplifies these management tasks for a broad range of applications in a variety of execution environments. Ultimately, we hope to eliminate the need for customized management scripts for deploying, running, and monitoring distributed applications in any wide-area networked environment.

Plush to the Rescue

Our solution to the problem is a system called Plush. Plush is a distributed application management infrastructure that leverages the insight that there are many similarities in the common tasks provided by customized management scripts. In particular, the first step is typically to locate and configure resources capable of hosting the application. After the resources are configured with the required software, the execution is started. Upon the completion of an execution, the scripts perform "cleanup" actions to ensure that the resources are left in a usable state for future executions. Rather than reinventing the same functionality repeatedly for each application and each execution environment, Plush automates these tasks and allows developers to define their application- and environment-specific details separately, making it easy to run applications in different distributed environments without rewriting or recreating customized scripts.

By eliminating the need for customized management scripts, Plush allows a wider range of developers to deploy and manage distributed applications running on hundreds of machines worldwide. Plush provides software developers with a user-friendly graphical user interface (GUI) called Nebula so that even novice developers can experiment with a distributed set of resources for hosting their applications. For more experienced developers, Plush also provides a command-line interface for managing distributed applications. Finally, for developers who wish to interact with the functionality of Plush from within a program or script, Plush exports an XML-RPC interface.

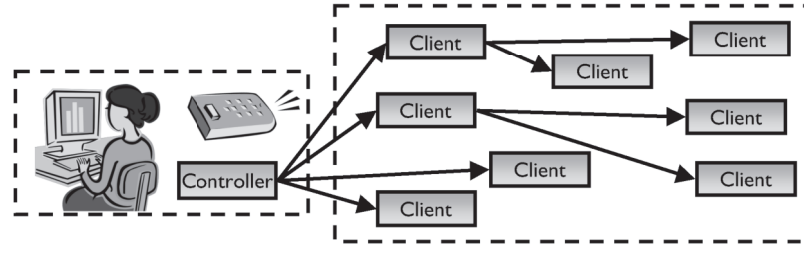

FIGURE 1: THE PLUSH CONTROLLER CONNECTS TO THE PLUSH CLIENTS RUNNING ON THE REMOTE RESOURCES.

Aside from the user interfaces, the architecture of Plush consists of two main components: the controller and the client. In most usage scenarios, the Plush controller runs locally and responds to input received from the software developer. The clients run on all remote resources involved in an application. In order to achieve scalability in Plush, the controller and clients build a communication tree for exchanging messages (Fig. 1). After establishing this tree, the controller communicates with the clients throughout the duration of an application's execution, both by sending instructions and by exchanging application management and status information. The Plush user interfaces interact directly with the controller, giving the developer a way to "remotely control" the resources hosting the application in a user-friendly way.

Plush in Action

To gain a better understanding of how Plush works, in this section we describe the tasks that Plush performs to manage a typical distributed application. These tasks are illustrated in Figure 2. More detailed information about the design and implementation of Plush can be found in our paper [1].

STEP 0: DESCRIBE THE APPLICATION

Before Plush can manage an application, the developer must provide the Plush controller with a description of the application and the desired resources for hosting the application. Typically, this is accomplished by creating a Plush application specification. When using Nebula (the Plush GUI), software developers create their application specification using a set of application "building blocks" that can be combined in an arbitrary fashion to define the custom control flow for their executions. There are separate blocks for describing resources and processes, so that developers are free to deploy applications on different resources without redefining any aspect of their execution. A resource in Plush is any computing device capable of connecting to the network and hosting an application. Developers use arrows connecting blocks to indicate the order in which various processes run within an execution. The right side of Figure 3 illustrates a sample application specification that uses the Plush building blocks. For command-line users, an XML file defines the application specification, which is loaded at the Plush prompt at startup.

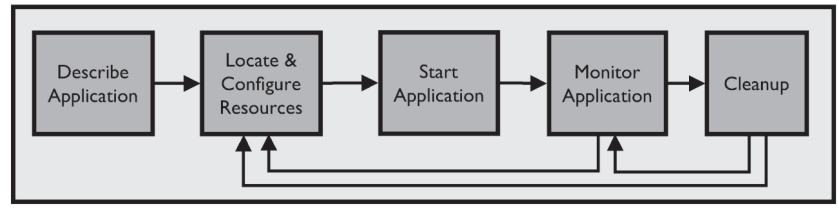

FIGURE 2: TASKS COMPLETED DURING A TYPICAL PLUSH MANAGED EXECUTION.

STEP 1: LOCATE AND CONFIGURE DESIRED RESOURCES

Once a developer creates an application specification, Plush has all of the information it needs to manage and configure a specific distributed application using a particular set of resources. From the developers' perspective, their job is done! At this point they can sit back and let Plush assume control of the application. After parsing the application specification provided by the developer, the Plush controller begins to locate and configure the desired resources. Depending on the target execution environment, this may involve using an external resource discovery service such as SWORD [7] to find resources with specific characteristics or creating new virtual machines for hosting the application with Shirako [5] or Usher [6]. Once the controller creates or locates the resources, the controller installs the Plush client software and then initiates a connection to each client. The clients' first task is to install the required software on the remote resources. Each client separately obtains the needed software packages and runs any necessary installation commands and then sends a message to the controller indicating that software installation is complete.

STEP 2: START THE APPLICATION

After the controller determines that a sufficient number of resources have been successfully configured, the controller instructs each client to start the

application's execution. The clients continue to inform the controller about application status changes throughout the duration of the execution. Some distributed applications operate in phases, where each resource involved in the application must compete a specific phase of execution before any resource proceeds on to the next phase. These applications typically require some form of distributed synchronization to ensure that the phases execute correctly across all resources. Plush provides support for a range of synchronization requirements in distributed applications [2]. In order to provide this synchronization, the controller maintains a list of each client's status at all times. The controller then determines when it is safe to allow an application to move on to the next phase of execution, and it instructs the clients accordingly. Therefore, not only does Plush manage the initial starting of the application, but it also ensures that multi-phased applications start each phase of execution at the correct time.

STEP 3: MONITOR THE APPLICATION'S EXECUTION

Detecting and recovering from failures is one of the most challenging aspects of running an application on resources spread around the world. To accomplish this in Plush, the clients running remotely monitor the status of the application and resources. If a client detects a problem, ranging from insufficient disk space to unexpected program termination, the client sends a message to the controller describing the failure. The controller then decides how to recover from the problem. Plush provides built-in mechanisms for recovering from many common failures, and in most cases, Plush is able to detect and recover from errors before the developer is even aware that a problem occurred. Some failures may require finding new resources for hosting the application, whereas others may only require restarting a failed process on a single resource. For more elaborate application-specific recovery, developers can use the Plush XML-RPC interface to implement their own failure-recovery routines and then register to receive callbacks from the controller when Plush detects failures. The GUI also lets users visualize their execution with color-coded dots on a map of the world (shown on the left side of Fig. 3), allowing them to easily monitor the status of their application by simply watching the dots change colors. Thus developers who use Plush no longer need to spend a significant portion of their time writing monitoring scripts and babysitting executions running on a distributed set of machines in order to keep their applications running.

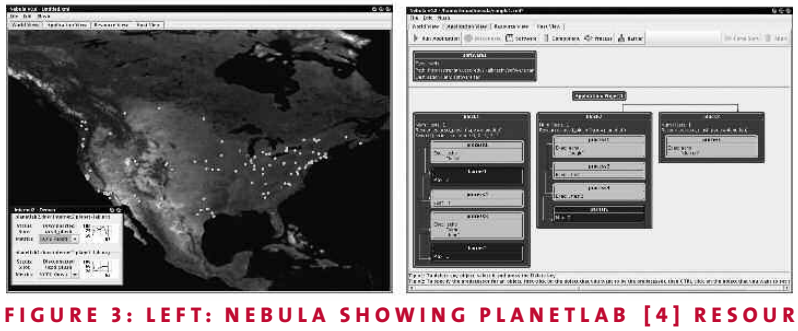

FIGURE 3: LEFT: NEBULA SHOWING PLANETLAB [4] RESOURCES RUNNING AN APPLICATION. RIGHT: AN APPLICATION SPECIFICATION BUILT USING PLUSH APPLICATION BUILDING BLOCKS.

STEP 4: CLEAN UP RESOURCES

The final task that Plush completes is to clean up the resources that host the application so that they are left in a usable state for future applications or future phases of execution. The cleanup procedure ensures that all processes exit cleanly, removing any unnecessary files and returning the state of each

resource to what it was before the execution began. In general, this procedure may run at any time during the application's execution, although in most applications it is typically only run between phases or at the completion of the execution. When the controller receives messages from all clients indicating that the execution has ended (or receives input from the developer indicating that the execution should be aborted), the controller instructs the clients to kill any remaining processes associated with the application. After killing all processes, the clients also remove any unnecessary files that were created as a result of the execution. Once all clients complete the cleanup actions, the controller instructs the clients either to continue with the next phase of execution or to disconnect from the Plush communication tree and stop the client process.

Performance Evaluation

To demonstrate how well Plush recovers from failures in a wide-area networked environment, Figure 4 evaluates Plush's ability to detect host failures and subsequently to find and configure replacement resources for SWORD running across PlanetLab. SWORD is a wide-area resource discovery service that requires each host to download and install a 38-MB software package before starting the execution. PlanetLab is a distributed execution environment consisting of 800+ resources spread across 40+ countries. In this experiment, Plush starts SWORD on 100 randomly selected PlanetLab machines, including some machines behind DSL network links. After 1250 seconds, we manually kill SWORD on 20 of the initial 100 machines to simulate host failures. The Plush clients independently notify the controller of the failures, and the controller locates and configures replacement resources for the ones that failed. The SWORD service is fully restored across 100 machines 1000 seconds later.

Using Plush to manage this application allowed us to avoid writing a custom script that probed for and recovered from host failures. Particularly for long-running services such as SWORD, developers need automated mechanisms for monitoring the behavior of the execution and coping with problems that arise. It is unrealistic to expect the developer of a service to constantly monitor its performance, but at the same time, a service must quickly and automatically recover from failures since other developers may rely on the functions that it provides. When using Plush, clients running on the PlanetLab resources monitor the service's performance at all times and automatically recover from failures. More details about this experiment are discussed in the paper by Albrecht et al. [1].

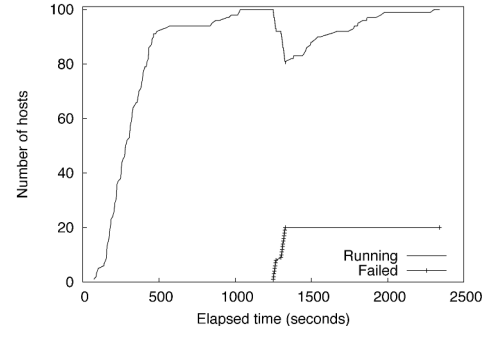

FIGURE 4: PLUSH INITIALLY STARTS SWORD ACROSS 100 PLANETLAB RESOURCES. AFTER 1250 SECONDS, 20 OF THESE RESOURCES FAIL. PLUSH AUTOMATICALLY DETECTS AND REPLACES THE FAILED HOSTS AND RESTORES THE APPLICATION.

How Do I Use Plush?

Plush is an open-source, publicly available software package that can be obtained from the Plush Web page [8]. Plush is implemented in C++, and it runs on most UNIX-based platforms. It depends on several C++ libraries, including those provided by xmlrpc-c, curl, xml2, zlib, math, openssl, readline, curses, boost, and pthreads. In addition, the command-line user interface requires packages for lex and yacc. (We typically use flex and bison.) If you intend to use Plush on PlanetLab, the controller uses several simple Perl scripts for interacting with the PlanetLab Central database. The only requirement related to network connectivity is that the Plush controller must be able to SSH to all remote resources.

Nebula is also publicly available. Nebula is implemented in Java and runs on any platform that supports Java, including most UNIX-based platforms, Windows, and Mac OS X, among others. Nebula communicates with the Plush controller using the XML-RPC programmatic interface. XML-RPC is implemented in Nebula using the Apache XML-RPC client and server packages. One additional benefit of using Nebula is that because it communicates with the Plush controller solely via XML-RPC, it is not necessary to run Nebula and the Plush controller on the same machine. If Nebula and Plush run on separate machines, after starting Nebula locally, developers have the option, using the Nebula preference menu, of specifying a Plush controller process running remotely.

Plush is currently in daily use worldwide. We have used Plush to successfully manage a variety of distributed applications, ranging from long-running services to short-lived, multi-phased computations. These applications were run in several different resource environments, including PlanetLab, ModelNet [9], and clusters of Xen [3] virtual machines. Although user feedback thus far has been largely positive, our goal is to make Nebula and Plush as user-friendly as possible, so we welcome all comments, suggestions, and feedback. If you would like further information, please visit our Web site (http://plush.cs.williams.edu), and feel free to contact any of the authors if you have additional questions.

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," *Proceedings of the USENIX Large Installation System Administration Conference (LISA)*, 2007.

[2] J. Albrecht, C. Tuttle, A.C. Snoeren, and A. Vahdat, "Loose Synchronization for Large-Scale Networked Systems," *Proceedings of the USENIX Annual Technical Conference (USENIX)*, 2006.

[3] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the Art of Virtualization," *Proceedings of the ACM Symposium on Operating System Principles (SOSP)*, 2003.

[4] A. Bavier, M. Bowman, B. Chun, D. Culler, S. Karlin, S. Muir, L. Peterson, T. Roscoe, T. Spalink, and M. Wawrzoniak, "Operating Systems Support for Planetary-Scale Network Services," *Proceedings of the ACM/USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2004.

[5] D. Irwin, J. Chase, L. Grit, A. Yumerefendi, D. Becker, and K. G. Yocum, "Sharing Networked Resources with Brokered Leases," *Proceedings of the USENIX Annual Technical Conference (USENIX)*, 2006.

[6] M. McNett, D. Gupta, A. Vahdat, and G.M. Voelker, "Usher: An Extensible Framework for Managing Clusters of Virtual Machines," *Proceedings of the USENIX Large Installation System Administration Conference (LISA)*, 2007.

[7] D. Oppenheimer, J. Albrecht, D. Patterson, and A. Vahdat, "Design and Implementation Tradeoffs for Wide-Area Resource Discovery," *Proceedings of the IEEE Symposium on High Performance Distributed Computing (HPDC)*, 2005.

[8] Plush Web page: http://plush.cs.williams.edu.

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