SPIE Newsroom

10.1117/2.1200803.1021

An integrated software environment for distributed systems development

Jeannie Albrecht, Ryan Braud, Charles Killian, Priya Mahadevan, Kashi Vishwanath, and Amin Vahdat

New tools simplify many tasks associated with designing, managing, and evaluating computer programs that run on thousands of computers worldwide.

As the number and diversity of Internet users continue to increase, Internet-based services—such as search engines and news websites—require more computing power in remote locations to satisfy user demand. Rather than hosting these services on a single centrally located server, companies are beginning to distribute the computing workload across thousands of servers worldwide. These distributed systems offer a number of advantages, including improved reliability and performance. However, designing and implementing a program that simultaneously runs on thousands of computers also introduces many new challenges to software developers, such as maintaining consistency across machines, evaluating performance, and detecting and recovering from errors.

To better understand these challenges, we now consider the tasks required to develop a distributed system. We group these tasks into three distinct phases: design and implementation, large-scale testing and evaluation, and wide-area deployment. The design and implementation phase involves writing code that accomplishes the goals of the target system. Next, the testing and evaluation phase looks for potential problems with the initial system design and corrects any bugs that may hinder performance. It is important to thoroughly test the code in a variety of settings that represent several different potential usage models of the system. After determining that the code behaves as expected and achieves high performance in the test cases, the final step is to deploy the program across the Internet and evaluate its performance. We have developed a number of tools that simplify the tasks involved with each of these phases, as described below. While individually valuable, together these tools form an integrated environment for building, testing, and deploying distributed systems.

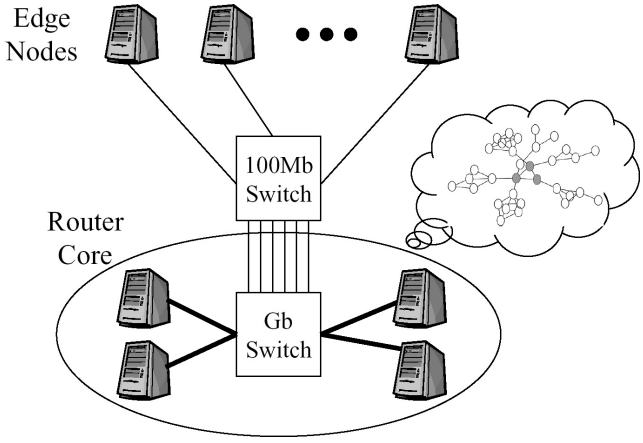

Figure 1. *The ModelNet emulator allows developers to evaluate distributed systems in realistic network environments without modifying their code.*

Phase 1: design and implementation

Since most distributed systems are designed to run on computers connected to the Internet, the code we develop must be robust enough to accommodate volatility. Internet-connected computers are often failure-prone, and wide-area network conditions tend to vary at high rates. Others have developed tools that help software developers address these issues. However, many are difficult to use correctly and often negatively impact system efficiency. We have developed the Mace[1] distributed systems toolkit, which overcomes these limitations by providing a powerful, event-based framework for dealing with both networking and event handling. Developers describe their systems using a simple but expressive specification template lan-

Continued on next page

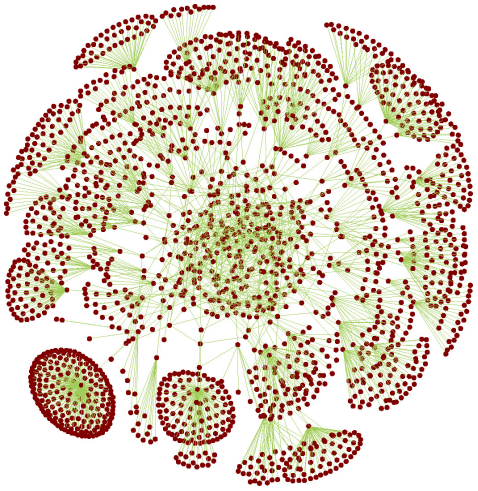

Figure 2. This 2000-node topology was generated by Orbis.

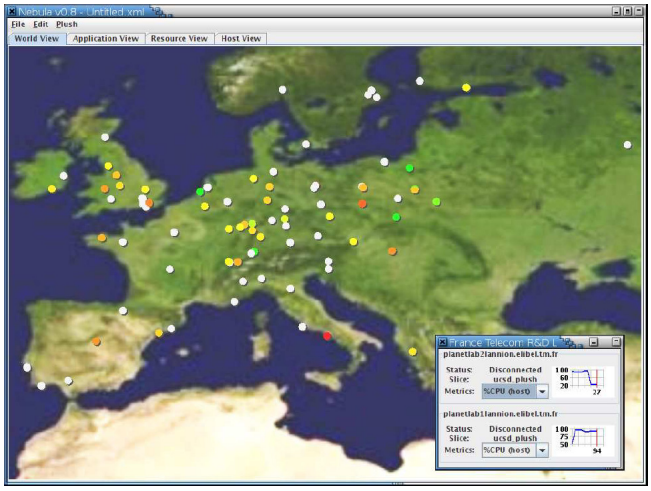

Figure 3. The Nebula interface to the Plush distributed system management framework displays the status of a program running on PlanetLab machines in Europe.

guage, augmented with C++ event handlers, that is then translated into a standard C++ implementation. Developers achieve performance by using a low-level programming language, and simplicity by allowing the compiler to write the tedious and repetitive code. Mace provides additional debugging tools,[2] including the Mace model checker (MaceMC),[3] which finds liveness bugs (violations of desired system behaviors) using a combination of systematic execution and long random executions.

Phase 2: large-scale testing and evaluation

After building the system, the next step is to perform large-scale tests in realistic conditions. The goal of this phase is to evaluate performance without actually running the code on the Internet, because many immeasurable and uncontrollable factors make it especially challenging to draw conclusions based on testing and evaluation. It is easier to perform initial tests in controlled environments, where developers can isolate and measure specific components of their programs separately. As a result, many developers resort to network simulators for testing their systems in large-scale settings. Simulators give developers complete control over their environment. However, simulators often require developers to modify their code, which could introduce new bugs or hide problems that exist in the actual code. In response to these limitations, we developed ModelNet[4] (see Figure 1), a large-scale network emulator that allows developers to evaluate distributed systems in realistic Internet-like environments without modifying the program code. ModelNet subjects the packets of unmodified applications to the hop-by-hop latencies, bandwidths, and queueing policies of a user-specified network topology in real time. Additionally, ModelNet can be used in conjunction with large-scale network topologies generated by Orbis[5] (see Figure 2) and realistic Internet traffic generated by Swing[6] to give developers a wide range of testing scenarios for their programs.

Phase 3: wide-area deployment

The unpredictability and volatility of the Internet often uncover a variety of new problems and bugs in programs. In the past, it was difficult to obtain access to hundreds of machines worldwide for testing purposes, and many developers were unable to complete this final phase of development. The recent introduction of testbeds such as PlanetLab[7] now give developers an opportunity to test their distributed systems on hundreds of computers spread across the Internet. PlanetLab machines run Linux, and are accessible only via SSH (secure shell). Developers who wish to use PlanetLab resources often spend a significant amount of time configuring the PlanetLab machines for their program and then monitoring the execution in an effort to detect and recover from errors. Plush[8] is a distributed system management framework that automates many of these tasks and simplifies error detection and recovery. Plush provides several different user interfaces for interacting with programs running across PlanetLab, including Nebula (shown in Figure 3), a graphical user interface that allows users to visualize the status of their program's execution. To help find the best set of PlanetLab machines available for running distributed systems, Plush uses remote procedure calls implemented via XML-RPC to interface

Continued on next page

directly with resource management services such as SWORD, a scalable wide-area resource discovery service for PlanetLab.[9]

Conclusions and future work

The state of the art in building, deploying, visualizing, and debugging distributed systems has not advanced much in the past 20 years. In many cases, researchers and software developers must still use customized, brittle scripts for managing distributed environments, and simple `printf` logging techniques for manual program inspection and debugging. The goal of our work is to build an integrated environment to ease this process. In particular, we have developed tools to simplify the three key phases of distributed systems development, including a programming toolkit for building and debugging code, a network emulator for advanced evaluation in controlled environments, and a management and visualization framework for Internet deployment and analysis. Moving forward, we are continuing to work on building additional language tools to aid with performance debugging of distributed systems, improving the realism of our evaluation environment, and enhancing the usability and functionality of our management framework.

Author Information

Jeannie Albrecht
Williams College
Williamstown, MA
http://www.cs.williams.edu/∼jeannie

Jeannie Albrecht is an assistant professor of computer science. She received her MS from Duke University in 2003, and her PhD from the University of California, San Diego (UCSD) in 2007 under the direction of Amin Vahdat and Alex C. Snoeren.

Ryan Braud, Charles Killian, Kashi Vishwanath, and Amin Vahdat
University of California, San Diego
La Jolla, CA
http://www.cs.ucsd.edu/∼rbraud
http://www.cs.ucsd.edu/∼ckillian
http://www.cs.ucsd.edu/∼kvishwanath
http://www.cs.ucsd.edu/∼vahdat

Ryan Braud is a PhD student working under the direction of Amin Vahdat in the Systems and Networking research group.

Charles Killian is a PhD candidate in the Department of Computer Science and Engineering under the supervision of Amin Vahdat. He completed his MS in computer science from Duke University in 2004. He expects to complete his PhD in June 2008.

Kashi Vishwanath is a PhD candidate. He works in the Systems and Networking research group under the direction of Amin Vahdat. Kashi expects to complete his PhD in 2008.

Amin Vahdat is a professor in the Department of Computer Science and Engineering and director of the Center for Networked Systems. He received his PhD in computer science from the University of California, Berkeley in 1998 under the supervision of Thomas Anderson. Before joining UCSD in January 2004, he was on the faculty at Duke University from 1999 to 2003.

Priya Mahadevan
Hewlett-Packard (HP) Labs
Palo Alto, CA
http://sysnet.ucsd.edu/∼pmahadevan

Priya Mahadevan graduated with a PhD in computer science from UCSD in 2007. She now works at HP Labs in Palo Alto. She completed her MS in computer science from Duke University in 2003.

References

1. C. Killian, J. W. Anderson, R. Braud, R. Jhala, and A. Vahdat, *Mace: language support for building distributed systems*, **Proc. Program. Lang. Design Implem.**, 2007. http://mace.ucsd.edu (accessed 21 Mar 2008)
2. P. Reynolds, J. L. Wiener, J. C. Mogul, M. A. Shah, C. Killian, and A. Vahdat, *Pip: detecting the unexpected in distributed systems*, **Proc. 3rd ACM/USENIX NSDI Symp.**, 2006. http://issg.cs.duke.edu/pip/ (accessed 21 Mar 2008)
3. C. Killian, J. W. Anderson, R. Braud, R. Jhala, and A. Vahdat, *Life, death, and the critical transition: finding liveness bugs in systems code*, **Proc. 4th ACM/USENIX NSDI Symp.**, 2007. http://mace.ucsd.edu (accessed 21 Mar 2008)
4. A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostic, J. Chase, and D. Becker, *Scalability and accuracy in a large-scale network emulator*, **Proc. 5th USENIX OSDI Symp.**, 2002. http://modelnet.ucsd.edu (accessed 21 Mar 2008)
5. P. Mahadevan, C. Hubble, B. Huffaker, D. Krioukov, and A. Vahdat, *Orbis: rescaling degree correlations to generate annotated Internet topologies*, **Proc. ACM SIGCOMM Conf.**, 2007. http://orbis.ucsd.edu (accessed 21 Mar 2008)
6. K. Vishwanath and A. Vahdat, *Swing: generating realistic packet traces*, **Proc. ACM SIGCOMM Conf.**, 2006.
7. L. Peterson, A. Bavier, M. E. Fiuczynski, and S. Muir, *Experiences building PlanetLab*, **Proc. 7th USENIX OSDI Symp.**, 2006. http://www.planet-lab.org (accessed 21 Mar 2008)
8. 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*, **Proc. 21st USENIX LISA Conf.**, 2007. http://plush.cs.williams.edu (accessed 21 Mar 2008)
9. D. Oppenheimer, J. Albrecht, D. Patterson, and A. Vahdat, *Design and implementation tradeoffs for wide-area resource discovery*, **Proc. IEEE Symp. High Perf. Distrib. Comp.**, 2005. http://sword.cs.wiliams.edu (accessed 21 Mar 2008)