A revised version of this paper was published in *Proceedings of the 2006 USENIX Annual Technical Conference*, Boston, MA, May–June 2006. Please read and cite the published USENIX 2006 paper in preference to this report.

Integrated Scientific Workflow Management for the Emulab Network Testbed

Eric Eide Leigh Stoller Tim Stack Juliana Freire Jay Lepreau

University of Utah, School of Computing
{eeide, stoller, stack, juliana, lepreau}@cs.utah.edu www.emulab.net

Flux Technical Note FTN–2006–01
February 2, 2006

Abstract

The main forces that shaped current network testbeds were the needs for realism and scale. Now that several testbeds support large and complex experiments, management of experimentation processes and results has become more difficult and a barrier to high-quality systems research. The popularity of network testbeds means that new tools for managing experiment workflows, addressing the ready-made base of testbed users, can have important and significant impacts.

We are now evolving Emulab, our large and popular network testbed, to support experiments that are organized around scientific workflows. This paper summarizes the opportunities in this area, the new approaches we are taking, our implementation in progress, and the challenges in adapting scientific workflow concepts for testbed-based research. With our system, we expect to demonstrate that network testbeds with integrated scientific workflow management can be an important tool to aid research in networking and distributed systems.

1. Introduction

In the networking and distributed systems communities, research and development projects are increasingly dependent on evaluating real applications on *network testbeds*: sets of computers, devices, and other resources that either emulate real-world networks or are overlaid upon the real Internet. Data from testbed experiments are increasingly important as software systems become larger and more complex. Many testbeds are now available to researchers and educators worldwide, including PlanetLab, RON, the Open Network Laboratory, ORBIT, and Emulab [15].

Our research group has been developing and operating *Emulab*—a free-for-use, Web-accessible, time- and space-shared, reconfigurable network testbed—since April 2000. Emulab provides a common interface to a dozen types of networked devices, unifying them within a single experimental framework. These devices include hundreds of (wired) PCs, a variety of wireless devices including PCs and motes, PlanetLab and RON machines scattered around the Internet, and emulated resources such as network conditioners and virtual machines. Today, our testbed has over 1,200 users from more than 170 institutions around the globe, and these users ran over 18,000 experiments in the last 12 months. Dozens of papers have been published based on experiments done with Emulab, and over 20 classes have used Emulab for coursework. In addition, Emulab's software is now operating more than a dozen other testbed sites around the world.

Emulab has continually grown to support larger and more varied experiments, and concurrently, users have applied Emulab to increasingly sophisticated studies. As a result, we have found that we must evolve our testbed facilities and interfaces to support more structured experiments. The current interface to Emulab is based on a notion of isolated experiments: an experiment is essentially a description of a network topology that can be instantiated (emulated) on the testbed. Many users have outgrown this model, however, and need better ways to organize, record, and analyze their work. Therefore, to support increasingly sophisticated research, we are evolving Emulab to support and manage scientific workflows *within* and *across* experiments.

A Different Domain, A Different Approach

We are inspired by the many scientific workflow management systems that have been developed for computational science in the Grid, including Kepler [9], Taverna [11], Triana [14] and others [16]. None of these systems directly apply to Emulab, however, because Emulab is about using unabstracted network resources and its model of use is different, as described below.

This material is based upon work supported by NSF under grants CNS–0524096, CNS–0335296, and CNS–0205702.

Furthermore, it is fair to say that "general purpose" scientific workflow systems have been slow to catch on and are not yet in broad use [8, 12]. Anecdotal evidence indicates that resistance to their adoption stems from two main factors [8, 12]. First, a workflow system may be too constraining: the system does not match its users' needs in some manner, so users subvert the environment. Once workflow escapes the tool, a substantial part of the tool's benefits are lost. Second, the intended audience may lack "buy in." People are reluctant to change their habits to use a new tool, especially if it requires them to change their work patterns. The impedance mismatch can be large or small, but even a small mismatch is significant. As traditionally approached and implemented, workflow management is about imposing discipline in ways that require active user support. Our experience with Emulab is that testbed users perform both structured *and* unstructured experiments, and that Emulab is most successful when it does *not* require special actions from its users.

Because we have tight control over both the testbed resources and the workflow user interface, we can do "better" than Grid workflow systems for our domain. To be successful, a workflow system must carefully fit into and support existing users' modes of use and the experiment life cycle, as the Taverna group describes [11]. We have experience with how people use Emulab and are crafting our workflow system to match. The following are some important ways in which our work is distinguished from other scientific workflow efforts, and we believe that these qualities will help us succeed.

- **Domain and Expertise.** We are tailoring our system to networking and distributed systems activities. That is precisely *our primary domain of expertise*, which we understand expertly. We have systems skills, helping us both to conceive and to execute techniques that most developers of workflow systems would not.

- **Experiment Model.** Emulab already has a pervasive *experiment* abstraction, which is a container of resources, control, and naming scope (Section 2). This abstraction eases both implementation and adoption: users are already familiar with it.

- **Implicit vs. Explicit Specification.** As an example of how our systems abilities have benefited our workflow system, it is easy for us to modify operating systems in the testbed and monitor the network to determine filesystem reads and writes. This helps us overcome the user-adoption barrier in which the experimenter must exercise discipline by explicitly specifying all input and output parameters. We can instead *infer* that open files are input or output parameters, and simply record all file state, or provide users with straightforward options to deal with them (Section 3).

- **History-Based Views.** A key part of our philosophy is to "remember everything." We can leverage the trend in cheap disk space and copy-on-write filesystem techniques to (in principle) record everything that occurs within an experiment. One possible implementation technique is to use a filesystem that supports snapshots, such as Sun's ZFS filesystem [13], to provide immutable storage efficiently. We provide a simple GUI for users to easily peruse and branch the "tree" of experiment history (Section 4).

- **Incremental Adoption.** Systems and network researchers and students already make heavy use of the Emulab testbed. We build on that to transparently add features under the covers, lure users to exploit them, and get explicit and implicit user feedback. Emulab's interface is Web-based, so like all ASPs we can track experimenters' use of features and their paths through our interface.

- **Pragmatic Approach.** Finally, we have adopted a thoroughly practical attitude toward the incorporation of scientific workflow into the user experience of our testbed. As we incorporate features to support workflows within and across testbed experiments, we strive to maintain high degrees of transparency, flexibility, and extensibility. Our goal is for users to be able to choose the degree to which they want to interact with workflow features, and change their minds as their activities require.

We are hopeful that the techniques we develop and the experience we gain will be applicable to other domains beyond the systems and network areas.

Contributions

Our work addresses a convergence of demand and opportunity, and Emulab is a uniquely powerful environment in which a workflow system can succeed. The testbed provides leverage such as the ability to repeat distributed system experiments "exactly" and collect data automatically. We are using *implicit* and *automatic* techniques for workflow definition and execution where possible, and monitoring for *completeness*. We have much history on Emulab's use and are guided by actual user experience (including our own) which shows the need for *flexible* user interactions and patterns for workflow *history* and *groups*. We are deploying features in an *evolutionary* way, thus helping users and enabling feedback. Maintaining a *low barrier to entry* is a major goal of our work. The rest of this paper describes the opportunities and challenges ahead, and summarizes the status of our workflow-integrated Emulab.

2. Testbed Experiments

In the current Emulab, the main conceptual entity is an *experiment*. Primarily, an experiment describes a network: a set of computers and other devices, the network links between them, and the configuration of those devices and links. An experiment defines such things as the hardware type of each node in the network (e.g., a 3 GHz PC or a mobile robot), the operating systems and software packages to be loaded on those nodes, the characteristics of each network link (e.g., speed, latency, and loss), and the configuration of other entities such as network traffic generators. A user describes these items in an extension of the *ns* language [7], or alternatively, through a GUI in Emulab's Web interface. Either way, the user submits the description to Emulab and gives it a name. Emulab parses the description into an internal form and stores the data in its database.

At this point, the experiment can be "swapped in." When a user directs Emulab to swap in an experiment, Emulab maps the logical topology of the experiment onto actual testbed hardware, loads the requested operating systems and software onto the allocated devices, creates the network links (using VLANs), and so on.

When swap-in is complete, the user can login to the allocated machines and do his or her work. In contrast to swap-in, which is handled by Emulab, the organization and execution of the actual experiment steps are mostly left up to the user. Many users perform their experiments interactively, via remote shell access to the allocated computers. More sophisticated users, however, often automate some or all of the steps. Emulab provides some tools for this—e.g., an event service—but users must provide the higher-level frameworks and orchestration that make use of Emulab's built-in tools.

A user's experiment may last from a few minutes to many weeks, allowing the user time to run multiple tests, change software and parameters, or do long-term data gathering. When the user is done, he or she tells Emulab to "swap out" the experiment, releasing the experiment's resources. The experiment itself remains in Emulab's database so it can be swapped in again later.

3. Workflow Within Experiments

It has become clear that Emulab's experiment notion must change to meet the needs of increasingly complex experimental activities. Three critical requirements are *encapsulation*, *orchestration*, and *data management*, and satisfying these needs is the job of a scientific workflow system. Consequently, we are now evolving Emulab's interfaces and internals to support experiments built around workflows. Organizing Emulab experiments around existing scientific workflow metaphors and mechanisms is not straightforward. Here, we outline some of the challenges and the ways we are extending existing workflow notions to meet our users' needs.

Encapsulation. An essential part of a workflow system is keeping track of artifacts, i.e., experiment inputs and outputs. Unfortunately, Emulab's current model of an experiment does not encapsulate all the artifacts that are part of the experimental process. Some constituents, such as the software packages installed at swap-in, are referenced by name only. The experiment definition does not contain the software, but instead contains only a file name. Other objects including command scripts, program input files, and especially program output files, may not be referenced at all in an experiment description. Users simply know where these objects are located in Emulab's file system.

A critical first step toward a workflow-enabled testbed, therefore, is to evolve Emulab's representation of experiments to encapsulate all the elements—insofar as possible—that make up an experiment. This evolution has many technical aspects, and Emulab already provides a solid base for addressing them. For instance, Emulab today encapsulates disk images [6], which contain operating systems and other software. These are currently referenced from experiments by name, but it will be "straightforward" to keep disk images within an expanded type of experiment.

The more challenging aspects of encapsulation concern the user interface, and the effort required for a user to create an encapsulated experiment. It is important for the set of experiment constituents to be *complete*, so an experiment can be repeated and modified; it must also be *precise* (without extraneous detail), so the experiment can be stored, analyzed by the testbed, and understood by people. Because even a small experiment may read and/or write hundreds of files, it would be unworkable for Emulab to require users to identify all the parts of an experiment by hand. Therefore, we are implementing *automatic approaches* to determining the "extent" of experiments. For instance, we have modified Emulab to snapshot all the files referenced by an experiment, every time the experiment is swapped in or out. Snapshots are persistent, and thus provide a sound basis for encapsulating experiments and tracking revisions over time. To find files that may not be referenced by an experiment description (e.g., output files), we have implemented dynamic monitoring of filesystem accesses. Dynamic techniques may not be precise, however, in the sense that they may capture irrelevant detail. As our work evolves, we expect to combine automatic and manual techniques for defining experiments precisely: automatic for producing candidates and checking encapsulation, and manual for precision, classification, and annotation.

Orchestration. Emulab is a platform for directed experimentation, but just as important, it is a platform for open exploration. Emulab configures computers and networks very quickly (within minutes) and provides users with interactive access and total control over their machines. For these reasons, Emulab is often used for ad hoc activities including software development and debugging, platform testing (e.g., with many different operating systems), and live demonstrations. Therefore, a major focus of our workflow integration is to accommodate the wide variety of ways in which Emulab is used: for directed and exploratory work, with interactive and scripted commands, and with workflows that blend and combine these modes. The practical issue is for the workflow facilities not to get in the way when they are not wanted. The research issue is how to meet that goal while also supporting the evolution of unplanned and/or interactive experiments into repeatable and/or scripted workflows. This capability will necessarily build on Emulab's ability to monitor essentially all activity within the testbed. Using an automated workflow in an interactive way is also essential: a user may want to execute the workflow up to a point, and then "take the wheel" to explore a new idea (Section 4).

By evolving the current experimentation model, we expect to lower the barrier to orchestrating Emulab experiments, expand the set of activities that can be coordinated, and provide a framework in which users can create new types of workflow steps. Currently, due to the effort required, relatively few Emulab users take the time to carefully script and package their experiments. By integrating a workflow system and libraries of workflow elements, we will make it easier to create experiments from reusable "parts." For example, we have prototyped a point-and-click interface that generates graphs from data stored in a broad range of formats. We will pursue similar ideas, based on programming-by-example, to make it easier for users to interactively develop and record their workflows.

Data Management. Pre-packaged workflow elements help us present other new capabilities to users as well: e.g., automated collection of experiment data (via probes), data analysis steps, and visualization steps. For these tasks, we and our collaborators are working to connect a *datapository* [2]—a network measurement data storage and analysis facility—to Emulab. Experiment measurements will be saved in the datapository's database. By structuring user interactions as workflows, we will be able to describe data analysis and visualization steps that can occur after the primary resources for an experiment run have been released, i.e., after "swap out." An important issue will be to expand Emulab's resource scheduling capabilities: e.g., to handle demands that change over the course of a workflow and to handle new types of resources such as CPU and I/O within the datapository.

4. Workflow Across Experiments

The thorough study of a system typically requires many experiments. For example, an Emulab user may want to study a distributed system across a variety of network sizes and topologies. Furthermore, as results are obtained from experiment runs, the user may change his or her plan for future experiments. Users routinely modify experiment definitions to explore promising new avenues as they are discovered—and then want to return to the original course, or explore yet another new direction. Thus, testbed users need to navigate easily through "experiment space" on both planned and unplanned courses. As opposed to managing workflow *within* a single experiment, this type of navigation corresponds to workflow *across* experiments. Workflow across experiments deals with flexible grouping, managing changes to experiment definitions, and navigating through artifacts across time.

Definition and Execution. An essential first step is separating the main temporal aspects of experiments: i.e., definition and execution. These aspects are currently intertwined: for example, an experiment is either swapped in or swapped out, and a user cannot run two copies of a single experiment at the same time. As we evolve experiments to be more encapsulating, we are adding the ability to track and distinguish results that are produced by each run of an experiment. This is similar to the separation of workflow definition and execution found in the VisTrails system [3]. Even if an experiment description does not change, its output may—for example, due to the impacts of real-world effects such as the state of the actual Internet. The many runs of a single experiment form a group, and Emulab will provide users with straightforward access to the members of such groups. For example, users will be able to see if and how the results of an experiment change over time.

Grouping. Beyond grouping the executions of a single, unchanging experiment, our experience is that users need much more sophisticated grouping facilities. Most importantly, users need flexible ways to manage groups of experiment *definitions* in addition to groups of *executions*. A user who wants to run a series of related experiments today must do so in an ad hoc manner: by creating many similar but “unrelated” experiments, by modifying one experiment many times, or by combining all the tests into a single Emulab experiment. All of these approaches, however, disguise the essence of the series from Emulab. Support for workflows over groups of experiments, therefore, will require Emulab to have a built-in notion of experiment series. We are currently prototyping support for experiment series in Emulab, which involves two main challenge areas.

First, it is essential for “series” to be a highly flexible concept, allowing users to define and navigate through arbitrary groups of experiments. Users will want to group experiment runs in many ways, none of them necessarily being inherent or dominant. One experiment may belong to many groups at the same time: for instance, a test of a Web server may belong to a series in which the client load varies (requests/sec) and to one in which the server configuration varies (e.g., number of threads, or caching strategy). Users will want to define series by extension (i.e., particular configuration values), intension (iteration within a parameter space), and combinations of these. New groupings will be created after experiments are run, and parameters may be added or removed over time. All of these things mean that we must develop a highly flexible means for cataloging and tracking experiment executions.

Second, we will develop new scheduling algorithms for experiment runs. As hinted above, we will allow experiment definitions to have explicit parameters. Like a subroutine, the parameter values are provided when an experiment is executed; a series can therefore be a *generator* of experiments to be run. By creating workflows that move though the parameter space of an experiment, Emulab will provide a scalable way to execute large numbers of trials. However, this presents a scheduling issue. Series and other groupings lead to new concerns such as the desire to get “interesting” results quickly. They also introduce new opportunities, including workflows that are “self-directed” to find interesting configurations. Our work in this area will build on adaptation features that are already in Emulab [5]. The need for smart scheduling highlights the need for workflow to be *integrated* with the testbed, not merely built atop existing interfaces.

History. User-defined experiment groups support “spatial” navigation across related experiments. Just as important, however, is temporal navigation over experiments, singly or in groups. As previously described, users routinely modify their experiments over time to explore both planned and unplanned directions of research. We have found that it is important for users to

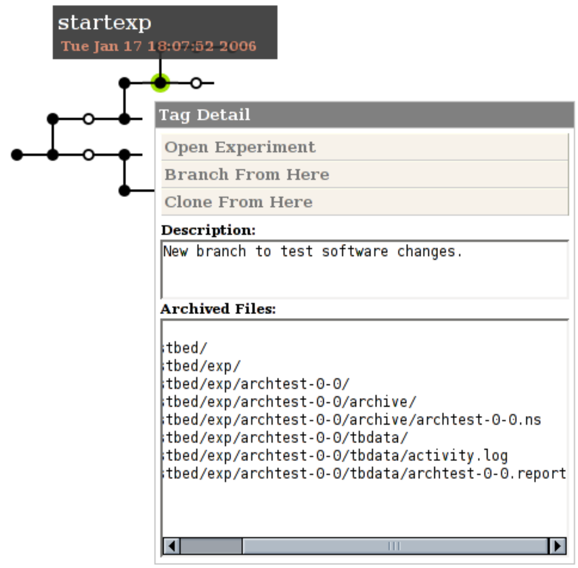

Figure 1. Screen shot of our prototype Web-based browser/editor of the experiment tree archive.

be free to make such changes without fear of losing previous work; to name and annotate experiment versions; and to “back up” to previous versions of an experiment, and from there “fork” to explore new avenues.

The revisions of an experiment form a tree, which represents the provenance of an experiment workflow. We have implemented a prototype of experiment trees in Emulab, based on the Subversion revision control system and an AJAX-based GUI, integrated into Emulab’s Web interface (Figure 1). Previous work by Bavoli et al. [3] and a paper to appear [4] provide additional detail about managing workflow histories in general.

5. Related Work

There is growing interest in applying scientific workflow systems to testbed-based research. For example, Plush [1] is a workflow-centered tool for experiments within PlanetLab. Plush focuses on tool integration and experiment automation; in contrast, our workflow enhancements to Emulab focus on strong encapsulation (including repeatability) and enhanced exploration (including flexible use, groups, and history management). Both Plush and our work aim to support extensions via third-party and user-provided workflow steps.

LabVIEW [10] is a popular commercial product for scientific experiment management and control. It is similar to our Emulab-based workflow system in that both manage workflow processes and results in the domain of computational science. However, LabVIEW differs in two important ways from our system and domain. First, the "instruments" in Emulab consume and produce many complex sources of data: e.g., configurations of hardware and software, input and output files and databases, software under test (sources and binaries), and data from previous runs and variations of an experiment. Emulab and its users must deal with a wide variety of highly structured data, not just series of sensor readings. Second, Emulab *is* the laboratory, not just the monitoring and control portion of it. Our integrated workflow system has near-total control over the "real world" in which Emulab experiments occur, so it can perform tasks such as setting up hardware and running experiments automatically, with probes in place. This is not generally available in a physical laboratory. Furthermore, to a significant extent, our system can archive the actual "devices" under test, not just the recorded outputs of those devices. The ability to archive and re-execute software—including entire disk images—means that we can provide much more automation in our domain than what LabVIEW can generally provide in its domain.

6. Conclusion

As grids changed the playing field for computational science, testbeds like PlanetLab and Emulab have changed the field for networked and distributed systems. The scale, complexity, and popularity of network testbeds have reached the point where scientific workflow systems are needed to manage testbed-based research. To meet this need, we are now integrating novel workflow support in Emulab, both within and across experiments. Moreover, testbeds like Emulab are an opportunity for advancing scientific workflow systems themselves. By building on and retaining Emulab's strengths, including "total" experiment monitoring and interactivity, we are expanding the domain of scientific workflow, developing new workflow and experiment management techniques, and, we predict, achieving new levels of acceptance and adoption.

Acknowledgments

We thank Mike Hibler and Russ Fish for their valuable comments on earlier drafts of this paper.

References

[1] J. Albrecht, C. Tuttle, A. C. Snoeren, and A. Vahdat. PlanetLab application management using Plush. *ACM Operating Systems Review*, 40(1), Jan. 2006. To appear.

[2] D. G. Andersen and N. Feamster. Challenges and opportunities in Internet data mining. Technical Report CMU–PDL–06–102, Carnegie Mellon University Parallel Data Laboratory, Jan. 2006. `http://www.datapository.net/`.

[3] L. Bavoli, S. P. Callahan, P. J. Crossno, J. Freire, C. E. Scheidegger, C. T. Silva, and H. T. Vo. VisTrails: Enabling interactive multiple-view visualizations. In *Proc. IEEE Visualization 2005*, pages 135–142, Oct. 2005.

[4] S. P. Callahan, J. Freire, E. Santos, C. E. Scheidegger, C. T. Silva, and H. T. Vo. Managing the evolution of dataflows with VisTrails. In *IEEE Workshop on Workflow and Data Flow for Scientific Applications (SciFlow 2006)*, Apr. 2006. Extended abstract. To appear.

[5] M. Hibler, R. Ricci, L. Stoller, J. Duerig, S. Guruprasad, T. Stack, K. Webb, and J. Lepreau. Feedback-directed virtualization techniques for scalable network experimentation. Flux Technical Note FTN–2004–02, University of Utah, May 2004. `http://www.cs.utah.edu/flux/papers/virt-ftn2004-02.pdf`.

[6] M. Hibler, L. Stoller, J. Lepreau, R. Ricci, and C. Barb. Fast, scalable disk imaging with Frisbee. In *Proc. 2003 USENIX Annual Tech. Conf.*, pages 283–296, June 2003.

[7] ISI, University of Southern California. The network simulator - ns-2. `http://www.isi.edu/nsnam/ns/`.

[8] G. Lindstrom. Personal communication, 2005–2006.

[9] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao. Scientific workflow management and the Kepler system. *Concurrency and Computation: Practice and Experience*, Dec. 2005. Accepted for publication.

[10] National Instruments. LabView home page. `http://www.ni.com/labview/`.

[11] T. Oinn, M. Greenwood, M. Addis, M. N. Alpdemir, J. Ferris, K. Glover, C. Goble, A. Goderis, D. Hull, D. Marvin, P. Li, P. Lord, M. R. Pocock, M. Senger, R. Stevens, A. Wipat, and C. Wroe. Taverna: Lessons in creating a workflow environment for the life sciences. *Concurrency and Computation: Practice and Experience*, Dec. 2005. Accepted for publication.

[12] S. Parker. Personal communication, Jan. 2006.

[13] Sun Microsystems, Inc. ZFS home page. `http://www.opensolaris.org/os/community/zfs/`.

[14] I. Taylor, I. Wang, M. Shields, and S. Majithia. Distributed computing with Triana on the Grid. *Concurrency and Computation: Practice and Experience*, 17(9):1197–1214, Aug. 2005.

[15] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. In *Proc. 5th Symp. on Operating Systems Design and Impl.*, pages 255–270, Dec. 2002.

[16] J. Yu and R. Buyya. A taxonomy of workflow management systems for Grid computing. Technical Report GRIDS–TR–2005–1, Grid Computing and Distributed Systems Laboratory, University of Melbourne, Australia, Mar. 2005.