Implementing the Emulab-PlanetLab Portal: Experience and Lessons Learned

Kirk Webb Mike Hibler Robert Ricci Austin Clements* Jay Lepreau

School of Computing, University of Utah

Abstract

Emulab's PlanetLab portal, hereafter known as the "portal," provides access to the large-scale, geographically distributed resources of the PlanetLab testbed using the integrated Emulab interface. The portal provides sophisticated resource allocation, configuration, and management services, while hiding from the user the underlying low-level detail and complexity of distributed resource provisioning and failure management. Moreover, it does so while minimizing the impact on the underlying PlanetLab system.

In the process of creating this portal and tracking PlanetLab's evolving third-party service API, we identified several key issues in the design of such platforms and the management systems built on top of them. This paper uses our portal as a basis for discussing these issues, and presents the lessons we have learned during its design and implementation.

1 Introduction

As the Emulab testbed [11] grows and evolves, much of its focus is on resource diversity and increasing experimental scale. Thus, we took serious note of the emergence of PlanetLab [2] as it offered the research community a significant set of new testbed resources. We wanted to both offer these new resources to existing Emulab users, and provide the PlanetLab community with a rich and powerful experiment setup and management interface.

One of PlanetLab's roles is to provide a substrate on which the PlanetLab user community can build multiple competing *infrastructure services* to manage and control PlanetLab resources. At its base, PlanetLab exports interfaces for creating *slices* and *slivers*. A sliver represents resources assigned to a user on a particular PlanetLab node, and a slice is a collection of slivers, spanning many or all PlanetLab nodes. Management systems build on top of these abstractions. The PlanetLab designers define a taxonomy of infrastructure services that may be constructed for PlanetLab [5]. Some tools have already emerged that provide a subset of these features, such as the PlanetLab-provided *Dynamic Slice Maintenance Tools*, the *Application Manager* [7], and *SWORD* [6]. The Emulab portal, in production since September 2003, is the first such service we know of that implements all parts of the taxonomy, and, in fact, adds some new ones. We give a brief overview of the implementation in Section 2.

We faced numerous challenges as we developed the portal, and we present here the lessons we learned as we dealt with them. From these lessons, we present recommendations for the designers and implementors of both the underlying infrastructure and value-added services on top of it. We believe that these lessons are applicable outside the PlanetLab context, in the broader scope of federating complex distributed systems such as these testbeds.

These challenges and their lessons are presented in Section 3. **Reconciling experiment models** describes the difficulty of preserving a model of short experiment life-cycles on top of APIs designed for longer cycles. Since both systems have their own idea of what resources exist, the state they are in, and how they have been allocated to users, **shared state management** is a difficult challenge that requires proper APIs and information sharing. The large, complex nature of systems like Emulab and PlanetLab, in conjunction with the potentially unreliable communication between them over the Internet means that failures are inevitable, and therefore careful **failure handling** is required. A final, ongoing problem is **working with interface evolution**.

2 Features

In contrast to PlanetLab's native experimental environment which is minimalist by design, Emulab provides an integrated "full-service" interface. Experiments are setup rapidly (on the order of a few minutes) and reliably, with the setup and subsequent control provided through Web, XML-RPC, or script-driven interfaces. Emulab experiments may be interactive or completely scripted and may be instantiated immediately or queued for setup when resources become available.

*Now at MIT; work done while at the University of Utah.

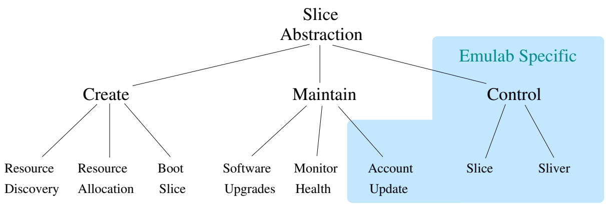

Figure 1: PlanetLab infrastructure service taxonomy adapted from [5]. Highlighted are services provided by the portal that are not part of the original taxonomy.

Nodes within an experiment are automatically setup with multiple user accounts, are pre-loaded with user-specified software, and start up user applications as needed. Dynamic control of nodes, and the experiment as a whole, is provided through an event mechanism. Activities such as link control, traffic generation, and program execution can be scheduled and synchronized via events.

All of these features, with the exception of the wide-area event mechanism which is still under development, are available in the Emulab-PlanetLab portal.

In the following paragraphs, we present the portal in terms of the aforementioned PlanetLab taxonomy, shown in adapted form in Figure 1. For clarity, we use "node" to refer to PlanetLab physical nodes and "virtual node" to refer to a node as seen by the Emulab infrastructure: i.e., a PlanetLab sliver loaded with Emulab software.

Resource discovery. The portal periodically queries PlanetLab Central (PLC) for a list of valid nodes, using this list to synchronize the Emulab database with PLC. Given the set of valid nodes, the portal can determine which are "live" and thus candidates for allocation to experiments. Liveness determination is accomplished by periodically instantiating an Emulab virtual node in a sliver on every PlanetLab physical node, rather than by relying on PlanetLab metrics or the success of underlying PlanetLab slice creation primitives. This "end-to-end" approach significantly reduces the observed failure rate of experiment setup.

For live nodes, we constantly monitor the available CPU, memory, disk and network resources to enable more accurate mapping of user experiments onto available resources. The monitoring is performed by a single, distinguished "service" sliver on every PlanetLab node.

Resource allocation. The portal offers three methods through which users allocate PlanetLab resources.

The most basic method is to manually choose individual nodes, as is done with PlanetLab's own current interface. This method also gives the user the opportunity to run their own selection algorithm before submitting their experiment specification to Emulab. Second, node selection can be done in a link-centric fashion. In this method, the user specifies a set of virtual nodes, and a set of virtual links between the nodes. Each virtual link can have a bandwidth, latency, and packet loss rate specified. Emulab then matches, as best it can, these desired link characteristics to end-to-end characteristics observed between PlanetLab nodes, gathered from third-party sensors. The third, and most common method of node selection is node-centric. In this scheme, users ask for a set of virtual nodes, with no links between them. For each virtual node, a type can be specified to restrict which PlanetLab nodes they may be associated with. For example, a user can ask for any node in PlanetLab, a node on a DSL line, or a node connected to Internet2.

Regardless of the resource specification technique used, the mapping of the desired user resources on to the available PlanetLab resources is done by Emulab's resource mapper, `assign` [9]. In addition to the criteria specified by the user above, `assign` also attempts to spread the requested nodes across sites, and to find nodes with low CPU and memory loads.

Emulab administrators can enforce admission control to PlanetLab by modifying system-wide parameters that determine the minimum per-node resources needed to instantiate any experiment. For example, they can indicate that only nodes with a load average below two and 10% of the disk free should be considered for allocation.

Boot(strap) slice. For every virtual node in an experiment, the portal uses a PlanetLab-provided hook to create a sliver on the appropriate PlanetLab node, load a standard set of Emulab scripts and tools, and trigger the standard Emulab "self-configuration" process. This configuration process sets up user accounts, downloads and installs user software packages, and fires up a user-specified experiment startup script to produce a working Emulab virtual node.

By default, if any of the PlanetLab or Emulab setup steps fail, the experiment setup fails. However, since reliable setup is not always possible or desirable in a true Internet environment, the user may specify that experiment setup succeed even if some virtual nodes fail to setup. For each virtual node that fails to setup, the PlanetLab node hosting that node is marked as no longer live and thus removed as a candidate for future experiments until the resource discovery mechanism determines that it is once again responsive.

Monitor health. One of the standard Emulab per-node services is a watchdog process that periodically sends a status message to Emulab Central. These messages are inputs to state machines tracking the "health" of every experiment. Failure to receive a status message from a virtual node for an extended period of time causes that node to be marked as "down." When the virtual node representing the service sliver on a PlanetLab node transitions to the down state, it is treated as if it had failed during initial setup, eventually causing instantiation to be attempted again. This is covered in more detail in Section 3. All other virtual nodes are associated with user experiments and currently, the portal does nothing more than record their status. Section 4 suggests a more aggressive response to failed experiment nodes.

Software upgrades and account updates. The reply to Emulab watchdog status message includes an indication as to whether a virtual node has outstanding account or software updates to perform. If an update is required, the watchdog contacts Emulab Central, downloads and installs new software, and updates accounts and ssh keys. Nodes check in for updates on the order of a few minutes.

Slice and sliver control. The portal provides an additional class of infrastructure services for dynamic control of slices (experiments) and slivers (virtual nodes). Currently this consists of being able to reboot individual slivers or an entire slice. Once deployed on PlanetLab nodes, the Emulab event mechanism will enable use of existing Emulab event agents to control traffic generation, program execution, and link testing.

3 Integration Challenges

Our work in developing the portal included handling a number of integration challenges. The underlying philosophies guiding the use models of both testbeds created contention in the setup phase. In addition, the distributed and shared state across the two loosely coupled systems often lost coherency. Finally, failure modes were common obstacles requiring continual tuning and attention. While working to address these issues, we made some observations on how large distributed systems could be adapted to ease integration.

Reconciling Experiment Models. Integrating Emulab's rapid-cycle experimentation model with PlanetLab's long-running, service-oriented model was nontrivial. Our goal was to extend support for using PlanetLab resources through Emulab while maintaining Emulab's successful use model. In Emulab's original cluster testbed, having local, reliable resources under our physical and administrative control allowed for very rapid experiment turnaround. When Emulab expanded to manage the RON testbed [1] and other distributed nodes, we faced the issues of unreliable, insecure communication and loss of physical control. However, the scale of distributed resources was small and we still maintained control of the infrastructure software, so preserving a rapid experiment life cycle was not a major effort.

However, with PlanetLab we must deal with a much larger set of resources that are externally controlled and allocate them via a more generic, and evolving, API. The original dynamic slice API, *dslice* [4], worked well under Emulab, allowing the portal to directly coordinate with individual PlanetLab node managers in a timely manner. The typical sliver instantiation time for a node under *dslice* was under 30 seconds. *dslice* mainly operated in a distributed manner, only requiring service integrators to contact a central server to obtain resource use tokens. All other communication was direct and synchronous to the individual node managers.

Along its evolutionary path, PlanetLab transitioned to a centralized management platform, the PlanetLab Central user interface (PLC, or PlanetLab version 2.0). Slice instantiation via PLC is asynchronous and provides no callback mechanism, or other explicit means to indicate completion. Thus, we were faced with trying to fit our fast, synchronous setup mechanics to the delayed, asynchronous semantics of PLC (which was targeted primarily at long-running services). We used per-sliver instantiation parallelization to hide much of the latency. For synchronization, we used both per-sliver polling via ssh, and the InstantiateSliver function that was added later at our request.

The PlanetLab Central user interface provides a hard one hour guarantee on sliver creation. However, given that the typical Emulab experiment lasts about five hours, we noted that this did not fit our use model, being fully 20% of experiment duration. We discovered that one out-of-band method for determining that sliver instantiation had completed earlier was to "poll" the sliver using ssh. However, we observed that *polling nodes for readiness is unnecessarily wasteful of resources.*

At our request, PlanetLab added a synchronous instantiation method, InstantiateSliver, to PLC. However, perhaps because it was not part of the original API design, it never worked well, proving to be unreliable (a 23% failure rate on nodes that had recently setup correctly) and exhibiting highly variable per-node instatiation times (from one to 15 minutes).

The more recent PLC/NM hybrid interface [8] (PlanetLab version 3) system includes a re-architected interface that exposes a lower level per-node manager, and like *dslice*, it does provide the fast, synchronous slice-level setup semantics that are a much better match for the Emulab system. Within this framework, it is also possible for a service provider to create a delayed, asynchronous sliver setup interface, such as the one provided by PlanetLab version 2. Therefore, *supplying lower level API primitives allows for a wider range of service models.*

Shared State Management. Managing shared state across Emulab and PlanetLab has been a difficult task, requiring ongoing refinements. Emulab maintains information about the resources users have requested and the environment they have asked to have set up on their nodes. Additionally, it gets information about available resources from PlanetLab Central and third-party sensors. Thus, node identity is an important issue, since there must be a consistent mapping between Emulab's and PlanetLab's notion of which nodes exist and what their state is. Identity is especially important in the PlanetLab context, in which the model is that many important services are provided by third parties rather than PlanetLab itself; each of these third parties must agree on a common resource identifier if they are to inter-operate.

As Emulab keeps records for all node resources it can allocate (physical and virtual), we need to keep our PlanetLab node records in sync with PlanetLab's. This is difficult because PlanetLab is a rapidly growing and changing platform; nodes are added and removed, and occasionally change identity. Hence, the portal must periodically poll PlanetLab for changes. Through this process, we found that there were no persistent identifiers available from PlanetLab to reliably distinguish its nodes. Any of IP, hostname, or internal ID number could and did change. To handle this, we implemented a matching heuristic to track changes. We also perform sanity checks to these changes by hand to ensure correctness. We recommend that PlanetLab *designate at least one persistent identifier to each node*, in order to simplify the task of identifying nodes. While defining the important identity characteristics of a node — i.e., its "role" — is difficult and subject to policy decisions, we think a consistent view and use of this distinguishing information is vital to service integrators, and therefore it should be pinned down. Then, *a node's unique identifier should change if and only if its role does* (which will hopefully be an infrequent event). In order to make state synchronization manageable, if a node is given a new identifier, it should appear from an allocation standpoint to be the same as a new node; that is, it should start with no resources allocated rather than holding over resource allocations from its old identity.

Emulab must also maintain state about slice expiration, node membership, and user permissions. Sanity checks between our lease records and PlanetLab's were included in the portal, and the Emulab experiment teardown phase was made robust against node and slice disappearance. Of particular note here is our experience with slice expiration. When the portal went through *dslice*, individual node slivers could expire independent of one another. To reduce overhead, we kept our own timers for each of these leases; this worked fine in practice, but was potentially dangerous as it assumed state synchronization. When we migrated to the PlanetLab Central user interface, we retained this loosely coupled synchronization model. After several slices went missing, due to disagreements between Emulab and PlanetLab's notion of the terms of their leases, we modified our renewal logic to synchronize with PlanetLab by periodically polling slice state via the PlanetLab Central user interface. In practice, a third party integrator should not rely on assumed state synchronization, especially over long periods of time. Software bugs, and policy or mechanism changes can easily invalidate a third party's information.

Resource state change callbacks are one mechanism that would help keep external service providers apprised of such changes. Exporting a complete view of the internal database structures and state machines would significantly augment the tools available for external state synchronization. Providing this information in a centralized fashion is advantageous, and a weakness in the original *dslice* model. However, distributed information could have been made available by publishing to a content distribution network, or via a peer-to-peer query substrate, e.g., Sophia [10].

Failure Handling. Given two large distributed systems such as Emulab and PlanetLab, failures are a fact of life and have many possible modes. Beyond the issues of large scale integration, both platforms have research aims, and so tend to evolve rapidly. This fact tends to produce instability in the interfaces. We have refactored the portal backend to more easily track new developments in PlanetLab, and have wrapped much of our remote interaction with robust failure handling mechanisms.

The portal backend defines wrapper functions that call a requested remote API function and handle different classes of error conditions encountered (continuable, fatal, and retryable). The classification of these errors is defined in software however; there is no heuristic to determine when to continue or give up. The portal backend also goes to great lengths to clean up and release all allocated resources in the event of failure. We also tolerate partial setup failure when bringing up an experiment across a PlanetLab slice. Any resources associated with nodes that fail to setup properly will be released.

One of the key observations we made in the course of this work is the *need for end-to-end testing*. Experience has shown that relying on the success of any subset of constituent PlanetLab or Emulab operations to indicate a node's fitness for use in an experiment isn't sufficient; one must perform a complete Emulab virtual node setup. There are such a large numbers of factors that can lead to failure, including network connectivity, disk space, node load, and software bugs, that no partial test can account for all failure modes. We have structured our service slice to perform this vital function in addition to its other duties. When a node fails to setup during experiment initialization, it is sent to a "down node" pool to await testing. This pool is also where newly discovered nodes first go to be tested for fitness. An Emulab daemon runs through these nodes in batches, executing a full setup of the service sliver on each one. Once setup is successful on a node, it is moved into the free pool where its resources become available for allocation to experimenters. Without this end-to-end testing, users would endure many more failures as Emulab attempted to setup their experiments on PlanetLab.

Working with Interface Evolution. A final challenge stems from the obvious tension between PlanetLab's requirements of "Evolving Architecture" which promotes evolution of the service-visible API and "Unbundled Management" which encourages third-party development of services [2]. As the architecture evolves, *backward compatibility should be maintained* so that existing management systems continue to work. The transition from the first PlanetLab interface to the second was abrupt, interrupting the availability and slowing the development of the portal. The transition to the third is occurring while the second is still available, making it much smoother. We attribute many of the problems we encountered initially in using PlanetLab Central user interface to the fact that the "native" PlanetLab interface does not interact with PlanetLab using the same API that it exports to external integrators. *We believe the best way to ensure that the externally available API is sufficiently powerful and robust is to use it internally as well.*

4 Future Work

We plan to enhance the portal to better support the long-running service use model on PlanetLab. Currently, nodes that fail to properly boot at setup time and nodes that fail after boot time, are lost to the experiment until the experiment is reinstantiated (via a swapout/swapin sequence or a modify operation). By implementing optional sliver state recovery, which will proactively reallocate and restart individual slivers, we can keep experiments fully instantiated. A related feature is an "all nodes" experiment that will attempt to create and maintain a slice encompassing all nodes in PlanetLab, automatically adding new nodes as they appear (and removing old ones as they disappear).

Other work will include finishing the extension of our event system for wide-area use. This will enable the same event-driven experimentation currently available in the Emulab cluster, including starting and stopping programs and controlling traffic generators.

We are starting to gather statistics on virtual node lifetime, to get hard numbers on end-to-end reliability.

Finally, we hope to start leveraging other infrastructure services built atop PlanetLab. These include the SWORD resource discovery service and distributed deployment mechanisms, such as CoDeploy [3], that will more efficiently scale up the amount of data we can move out to the vservers.

5 Conclusion

This paper describes our experience developing the portal, the associated challenges we faced in integrating two large, independent systems over the past year, and some of the lessons.

In particular, we have identified several integration challenges: reconciling different experimental models, such as rapid cycle, and long-running (service-oriented); managing state between loosely coupled service providers sharing common resources; handling failure modes in complex systems; and working with evolving APIs.

We feel that the lessons we learned in the course of meeting these challenges, as presented in this paper, will aid us, and hopefully others in future efforts to integrate large and potentially disparate testbeds.

Acknowledgments

For much implementation and information, we thank Brent Chun, Steve Muir, Larry Peterson, and Mic Bowman, as well as others involved in PlanetLab support. We thank NSF for their sponsorship under grants ANI–0082493 and CNS–0335296.

References

[1] D. Andersen, H. Balakrishnan, F. Kaaschoek, and R. Morris. Resilient Overlay Networks. In *Proc. of the 18th ACM Symposium on Operating Systems Principles*, pages 131–145, Banff, Canada, Mar. 2001.

[2] A. Bavier, M. Bowman, B. Chun, D. Culler, S. Karlin, S. Muir, L. Peterson, T. Roscoe, T. Spalink, and M. Wawrzoniak. Operating System Support for Planetary-Scale Services. In *Proc. of the First Symposium on Network Systems Design and Implementation*, Mar. 2004.

[3] CoDeploy: A Scalable Deployment Service for PlanetLab. http://codeen.cs.princeton.edu/codeploy/.

[4] dslice: A prototype implementation of a dynamic slice creation service for PlanetLab. http://dslice.planet-lab.org/.

[5] S. Karlin. An Overview of the PlanetLab Architecture, Jan. 2004. http://www.planet-lab.org/Talks/2004-01-27-JointTechs.pdf.

[6] D. Oppenheimer, J. Albrecht, D. Patterson, and A. Vahdat. Scalable Wide-Area Resource Discovery. Technical Report UCB//CSD-04-1334, UC Berkeley, 2004.

[7] PlanetLab Application Manager. http://appmanager.berkeley.intel-research.net/.

[8] PlanetLab NodeManager API. https://www.planet-lab.org/Wiki/bin/view/Planetlab/NodeManagerApi.

[9] R. Ricci, C. Alfeld, and J. Lepreau. A Solver for the Network Testbed Mapping Problem. *SIGCOMM Computer Communications Review*, 33(2), Apr. 2003.

[10] M. Wawrzoniak, L. Peterson, and T. Roscoe. Sophia: An Information Plane for Networked Systems. Technical Report PDN–03–014, PlanetLab Consortium, July 2003.

[11] 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. of the Fifth Symposium on Operating Systems Design and Implementation*, pages 255–270, Boston, MA, Dec. 2002.