Experience-Driven Experimental Systems Research*

Larry Peterson and Vivek S. Pai

Department of Computer Science
Princeton University

1 Introduction

PlanetLab is a global platform for experimentally evaluating wide-area network services [8, 9]. It currently includes over 800 nodes spanning 400 sites and 39 countries, and hosts over 600 experimental research projects investigating content distribution, anycast, anomaly and fault diagnosis, publish-subscribe, routing overlays, peer-to-peer networks, and so on. PlanetLab also helps motivate and demonstrate the feasibility of a research facility that catalyzes future Internet design [1], as exemplified by the National Science Foundation's Global Environment for Network Innovation (**http://www.geni.net**).

This makes PlanetLab an interesting experimental system at two levels: (1) it enables experimentation with new services at scale, and (2) PlanetLab is itself an experimental wide-area hosting service. The main lesson of our experience with PlanetLab and the services it hosts is that it is only through building, deploying, and *using* experimental systems that we understand the important issues that influence their design. Such *deployment studies* lead to systems that evolve incrementally based on experience gained from supporting a live user community. These studies complement experimental evaluations under controlled conditions using synthetic workloads. The latter permits the thorough evaluation of tradeoffs within a space defined by a set of assumptions. The former is necessary if we want to expose implicit assumptions, or discover the relevant set of assumptions in the first place.

While a discussion of the breadth of services that run on PlanetLab would undoubtedly be valuable,[1] this paper has a more modest goal. It explores the value of stressing experimental systems with real users in the context of two specific cases: PlanetLab itself, and the suite of services developed as part of the CoDeeN project (**http://codeen.cs.princeton.edu**). The paper is organized around three main themes: (1) using a system exposes implicit assumptions that lead to new research opportunities, (2) real systems often involve balancing competing objectives rather than optimizing along a single dimension, and (3) simple systems that scale "well enough" and are robust are better than more complicated systems that require fragile coordination mechanisms.

2 Uncover Research Opportunities

One of the most important lessons of PlanetLab is that making a system real and exposing it to users uncovers the next set of research opportunities that were not obvious at the outset. Supporting a continuously running

*A variant of this paper appeared in *Communications of the ACM*, November 2007.

[1]A partial bibliography of papers written by the network systems research community can be found at **http://www.planet-lab.org**.

system involves more work than running a prototype just long enough to get a performance number, but the cost is worth it from a research perspective. Our history building and measuring a content distribution network (CDN) illustrates this point.

Our research into CDNs began as an effort to design redirection strategies that would scale under load, yet have response times comparable to the best-known techniques. Using the best methodology of the day, we simulated our algorithms and quantified the potential improvement in aggregate throughput as 60-91% over the published state-of-the-art [12].

It is at this point that PlanetLab became operational, allowing us to implement a system that exploited the algorithms, and to deploy it in a realistic setting in hopes of better understanding the efficacy of our algorithms. However, within days of deploying the first version of the system—which we call CoDeeN—we learned an important lesson: unanticipated traffic (e.g., relaying spam) compromised the security of the system, making it unusable. We then augmented CoDeeN with new mechanisms that took this lesson and others into account [13]; subsequent experience led to improvements in this system [6].

Within weeks of deploying the new system, we discovered that performance was suffering due to failures of the Domain Name System (DNS). Based on additional observations, we determined the root cause to be unexpected failures of the client-side DNS servers, instead of the server-side DNS infrastructure that was the focus of most DNS research. In response, we demonstrated how the ideas in CoDeeN (which was designed to make web content more available) could be adapted to also make DNS resolution more robust. This resulted in the deployment of a companion system, call CoDNS [7], on PlanetLab.

By examining CoDeeN use over time, we next came to realize that users were benefiting not only from CoDeeN's caching of files, but also from CoDeeN's selection of network paths. The caching benefits were only being provided for smaller files, but large files were still being relayed (though not cached). However, by introducing a new set of mechanisms on top of CoDeeN, it was possible to split and collectively cache these large files, and scalably serve them to large numbers of clients, even for multi-gigabyte files. One of the more interesting lessons of this exercise is that many of the algorithms proposed by others to solve this problem do not work well in practice, and it is only by a thorough evaluation of various engineering tradeoffs that we were able to design a system (called CoBlitz) with robust performance under a wide range of conditions [5]. However, even this system went through successive refinements as we improved our understanding of large file usage, server behavior, and network congestion [2]. We say more about this experience in the last section.

In addition to discovering and solving the next set of problems, these running systems have also been a boon with respect to new data. For example, based on instrumentation of CoDNS, we discovered that many assumptions about DNS name stability and intelligent DNS redirection were wrong, and we produced data that can be used by other researchers [10]. We also collected data by instrumenting CoDeeN, and were able to observe orders of magnitude more Internet routing failures than any existing observation platform has yielded, resulting in a more accurate model of Internet failure behavior [15]. Follow-on work to more accurately identify these failures led to discoveries about how often router DNS names were wrong, and how even a small number of errors could lead to incorrect calculations of path inflation or ISP connectivity [14]. This research developed new heuristics to automatically identify and fix these problems in many scenarios, and will help improve the accuracy of network mapping and modeling. Finally, we recently adapted the CoDeeN network failure monitoring to be a continuously running service that reports Internet failures in real-time, which will help facilitate more network failure research, and allow other services and applications to adaptively route around failures.

An epilogue to this story is that we never bothered to return to the issue of the specific algorithms used in his original system, as they were in the noise relative to the other factors that actually influence an Internet service.

A second observation is that this story likely sounds familiar to developers in industry, where incrementally improving deployed systems is the norm. One could make the case that industry is best positioned to identify "real world" issues (with academia better positioned to frame problems and conduct controlled experiments), and in fact, one could even argue that much of what academic researchers learn through deployment studies is already known by their colleagues in industry. However, our experience suggests two counter-arguments. The first is that academic researchers are much more likely to pursue systems that do not (currently) have commercial value. These tend to be enabling and general-purpose systems (e.g., like PlanetLab) rather than systems that addresss an immediate customer need. The second is that deploying and instrumenting real systems gives researchers access to data that is not otherwise available. It is unfettered access to this data—and the ability to add instrumentation as needed—that spawns follow-on research.

3 Think Architecturally

Experimental systems research often focuses on evaluating engineering tradeoffs in an effort to improve one or two quantifiable metrics. In contrast, the design decisions that shaped PlanetLab—like many real-world systems that have to support real users—were in response to conflicting requirements. The result is a comprehensive architecture based more on balancing global considerations than improving performance along a single dimension. PlanetLab's design was guided by five major requirements.

1. It had to provide a global platform that supports both short-term experiments and long-running services that support a client workload.

2. It had to be available immediately, even though no one knew for sure what "it" is. This means we had to evolve PlanetLab incrementally.

3. We had to convince sites to host nodes running code written by unknown researchers from other organizations.

4. Sustaining growth depends on support for autonomy and decentralized control.

5. It had to scale to support many users with minimal resources.

PlanetLab supports the required usage model (Req 1) through *distributed virtualization*—each service runs in a *slice* of PlanetLab's global resources. Multiple slices run concurrently on PlanetLab, where slices act as network-wide containers that isolate services from each other.

To address the second requirement, PlanetLab adopted an organizing principle called *unbundled management*, which argues that the services used to manage PlanetLab should themselves be deployed like any other service, rather than bundled with the core system. The case for unbundled management has three arguments: (1) to allow the system to more easily evolve; (2) to permit third-party developers to build alternative services, enabling a software bazaar, rather than rely on a single development team with limited resources and creativity; and (3) to permit decentralized control over PlanetLab resources, and ultimately, over its evolution.

Beyond these two ideas, what made PlanetLab a challenge to design was dealing with conflicts among requirements. We do not discuss the details of how we resolved these conflicts here—the interested reader is refered to a companion paper [9]—but instead highlight the architectural features that we introduced to address each conflict.

The first conflict-induced issue was minimizing centralized components (Req 4) while maintaining the necessary trust assumptions (Req 3). We approached this by architecting PlanetLab's control plane to include a minimal trusted core. Both hosting sites and slice users depend on this trusted intermediary, but multiple such trusted entities can peer with each other (using a minimal interface) to support a federation of systems.

The second issue was isolating slices from each other (Req 1), yet allowing some slices to manage other slices for the sake of unbundled management (Req 2). To address this issue, PlanetLab includes a mechanism, called Proper, that grants slices narrowly defined privileged operations [3], thereby selectively "poking holes" in the isolation mechanism.

The third issue was balancing the need for slices to acquire the resources they need (Req 1), yet coping with scarce resources (Req 5). To address this, PlanetLab's architecture includes two features. First, it decouples slice creation from resource acquisition. This means slices hold resources only when required by a service or experiment (rather than the lifetime of the slice), and slices can use alternative resource allocators to acquire the resources they need. Second, PlanetLab depends on "fair share" resource allocation as its default mechanism, augmented with "recovery" mechanisms that deal with potential resource thrashing. One such mechanism kills the slice with the largest physical memory usage on a given node when that node's swap space is 90% utilized. This mechanism has given users an incentive to be careful about memory consumption, which has nearly eliminated memory as a bottleneck resource.

A non-technical observation is that the reward structure for experimental systems research is biased against architectural work. That is, researchers are more often rewarded for innovations that yield an x% improvement along some quantifiable dimension than they are for a synthesis of known techniques to produce a working system that balances conflicting requirements. Moreover, fostering an environment that encourages researchers to build and support continuously running services has long-term implications. Supporting such services provides an incentive to build general-purpose "helper" and "building block" services that are, in turn, of value to other researchers. The community then comes to depend on these sub-services, which means they must be maintained, resulting in an on-going funding challenge for the research community.

4 Keep it Simple

In many cases, we have adopted simple approaches as an initial stepping stone, only to find that they are more desirable long term than other approaches. In other cases, we have intentionally chosen simple approaches to gain robustness.

While most testbeds expect only one user at a time, PlanetLab was designed to allow multiple users sharing its resources, in order to support long-running distributed services. For simplicity, Linux was chosen as the per-node operating system, with an expectation of revisiting the decision later when virtual machines and other operating systems became more of an issue. In retrospect, choosing one operating system reduced management overheads, and allowed developing the kernel customization needed to scale with PlanetLab's growth. Nodes now regularly run 50-100 simultaneous experiments, in part enabled by the use of the VServer mechanism, which virtualizes the filesystem and userspace, while running only one OS kernel [11].

PlanetLab's popularity motivated the case for more control over resource allocation, with several competing proposals. We chose a simple approach, Sirius [9], that allowed any slice on PlanetLab to request a one-hour reservation of guaranteed CPU and link resources across PlanetLab. Conventional wisdom suggests such a scheme would lead to a tragedy of the commons, since there is no charge for the extra CPU. In practice, this mechanism works well, and is rarely even 50% subscribed. Moreover, the continuously running services on PlanetLab do not need this mechanism, since they all have their own adaptation mechanism to handle a wide range of node capacities, including heavily subscribed nodes. The short-term experiments will sometimes use this mechanism, but they tend to not run often enough to oversubscribe Sirius. As PlanetLab grows, we may look to enhancing Sirius, since it provides the boost across all PlanetLab nodes, and most of its users run on only a subset of nodes.

Simplicity has also guided the longest-running monitoring service on PlanetLab, CoMon [4], which centrally collects and analyzes node activity. While it may seem odd to not use a distributed system to monitor a distributed platform, the centralized approach has kept message traffic low, and reduces monitoring CPU utilization on the nodes. CoMon generally consumes about 0.3% of the node's CPU, and even monitoring 750 nodes every 5 minutes, it consumes less than 1 Mbps in aggregate. While the traffic consumption scales linearly with the number of nodes at the centralized collector, it stays constant at each node, which is important when monitoring nodes having network problems. This design also makes collecting certain statistics, such as median values and top values, much simpler.

One of the more interesting examples of simplicity in design is CoBlitz, our large file system, in which most design decisions run counter to conventional wisdom. It uses an unstructured topology, rather than building a distributed hash table or similar structure. While this approach causes more pairwise heartbeat traffic, it also has all of the liveness information traverse the same network paths as the data transfers, reducing routing problems caused by mismatched information. Having each node independently pick its own peers runs counter to ideas about scale, but we observe that even the largest commercial CDNs have on the order of 1000 PoPs. While running directly on end-user machines may sound appealing, security considerations have prevented HTTP-compatible systems from taking this approach. BitTorrent, in contrast, can have the server provide content hashes, which mitigates this risk, but CoBlitz is designed to run using standard Web servers and unmodified Web clients. This requirement has also driven us to using regular TCP connections, contrary to the finer granularity UDP provides for the current generation of DHTs. The incidental benefit of TCP, however, is that it also has fewer problems with stateful firewalls and intrusion detection systems. Despite all of these seemingly "unadvanced" design decisions, CoBlitz remains one of the busiest services on PlanetLab, and is the only large-file service that has stayed in operation. We attribute part of this fact to the observation that simplicity has made the system more robust, easier to debug, and easier to maintain over time.

While the systems community has long recognized the value of simplicity, it is equally true that there is constant pressure to "fix" simple systems to correct for their obvious flaws. Sometimes this is necessary if the system is going to "grow up" and be sustainable on a long-term basis. Sometimes it is merely to satisfy the need of a solution looking for a problem. The trick is to recognize the difference. The strategy with PlanetLab has been to architect the system so others can add enhancements—this happens through unbundled management and federation—which, in turn, depends on defining a minimal set of interfaces. We cannot claim to have gotten these interfaces right yet, but this is a central design principle of the effort.

5 Concluding Remarks

Classical science equates experimentation with running controlled experiments (lab experiments), which in Computer Science are often designed to evaluate implementation and engineering choices. However, Computer Science also benefits from deployment studies (field trials), which involve building and running prototypes that are subjected to real usage. This is because building something and watching it run helps us to identify implicit assumptions, the need for different functionality, surprising behavior, unexpected limitations, and so on. In this sense, such *experimental systems* work is like constructing a building—

engineering principles tell us whether it is sound design, but we need to build it and use it to decide how well it serves its purpose.

This paper illustrates how such studies proved beneficial for PlantLab and a suite of network services that run on top of it. Our observation is that deployment studies are valuable in exposing new research opportunities, that they force designers to think "architecturally" across the complete system rather than focus on a single dimension, and that real systems commonly benefit from simple designs, effectively discouraging the temptation to introduce complexity for complexity's sake.

Acknowlegements

This work was funded in part by NSF Grants CNS-0520053, CNS-0454278, and CNS-0335214, and DARPA contract N66001-05-8902.

References

[1] T. Anderson, L. Peterson, S. Shenker, and J. Turner. Overcoming the Internet Impasse Through Virtualization. *IEEE Computer*, 38(4):34–41, Apr 2005.

[2] B. Biskeborn, M. Golightly, K. Park, and V. S. Pai. (Re)Design Considerations for Scalable Large-File Content Distribution. In *Proc. 2nd WORLDS*, San Francisco, CA, Dec 2005.

[3] S. Muir, L. Peterson, M. Fiuczynski, J. Cappos, and J. Hartman. Privileged Operations in the PlanetLab Virtualised Environment. *SIGOPS Operating Systems Review*, 40(1):75–88, 2006.

[4] K. Park and V. Pai. CoMon: A Mostly-Scalable Monitoring System for PlanetLab. *ACM SIGOPS Operating Systems Review*, 40(1), Jan 2006.

[5] K. Park and V. S. Pai. Scale and Performance in the CoBlitz Large-File Distribution Service. In *Proc. 3rd NSDI*, San Jose, CA, May 2006.

[6] K. Park, V. S. Pai, K.-W. Lee, and S. Calo. Securing web service by automatic robot detection. In *Proceedings of the 2006 Usenix Annual Technical Conference*, Boston, MA, June 2006.

[7] K. Park, V. S. Pai, L. Peterson, and Z. Wang. CoDNS: Improving DNS Performance and Reliability via Cooperative Lookups. In *Proc. 6th OSDI*, pages 199–214, San Francisco, CA, Dec 2004.

[8] L. Peterson, T. Anderson, D. Culler, and T. Roscoe. A Blueprint for Introducing Disruptive Technology into the Internet. In *Proc. HotNets–I*, Princeton, NJ, Oct 2002.

[9] L. Peterson, A. Bavier, M. Fiuczynski, and S. Muir. Experiences Building PlanetLab. In *Proc. 7th OSDI*, Seattle, WA, Nov 2006.

[10] L. Poole and V. S. Pai. ConfiDNS: Leveraging Scale and History to Improve DNS Security. In *Proc. 3rd WORLDS*, Seattle, WA, November 2006.

[11] S. Soltesz, H. Potzl, M. Fiuczynski, A. Bavier, and L. Peterson. Container-based Operating System Virtualization: A Scalable, High-Performance Alternative to Hypervisors. In *Proc. EuroSys 2007*, Lisbon, Portugal, Mar 2007.

[12] L. Wang, V. Pai, and L. Peterson. The Effectiveness of Request Redirection on CDN Robustness. In *Proc. 5th OSDI*, pages 345–360, Boston, MA, Dec 2002.

[13] L. Wang, K. Park, R. Pang, V. S. Pai, and L. Peterson. Reliability and Security in the CoDeeN Content Distribution Network. In *Proc. USENIX '04*, Boston, MA, Jun 2004.

[14] M. Zhang, Y. Ruan, V. S. Pai, and J. Rexford. How DNS misnaming distorts internet topology mapping. In *Proceedings of the 2006 Usenix Annual Technical Conference*, Boston, MA, June 2006.

[15] M. Zhang, C. Zhang, V. S. Pai, L. Peterson, and R. Y. Wang. PlanetSeer: Internet Path Failure Monitoring and Characterization in Wide-Area Services. In *Proc. 6th OSDI*, pages 167–182, San Francisco, CA, Dec 2004.