By Larry Peterson *and* Vivek S. Pai

EXPERIENCE-DRIVEN EXPERIMENTAL SYSTEMS RESEARCH

The most direct way toward understanding whether PlanetLab and other such systems serve their purpose is to build, deploy, and use them.

PlanetLab is a global platform for experimentally evaluating wide-area network services [6]. As of September 2007, it included roughly 800 nodes spanning 400 sites and 39 countries and hosted more than 600 experimental research projects investigating content distribution, anycast, anomaly and fault diagnosis, publish-subscribe, routing overlays, and peer-to-peer networks. PlanetLab also helps motivate and demonstrate the feasibility of a research facility for catalyzing future Internet design [1], as exemplified by the National Science Foundation's Global Environment for Network Innovation (www.geni.net).

ILLUSTRATION BY PETER HOEY

This makes PlanetLab an interesting experimental system for new services at scale and as a wide-area hosting service. The main lesson from our experience with PlanetLab and the services it hosts is that it is only through building, deploying, and using experimental systems that we are able to fully understand the important issues that influence their design. Such deployment studies lead to systems that evolve incrementally based on experience gained from supporting an active and sizeable user community. These studies complement experimental evaluations under controlled conditions using synthetic workloads. Synthetic workloads permit the thorough evaluation of trade-offs within a systems domain defined by a set of assumptions. Support for an active user population 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 running on PlanetLab would be valuable,[1] our goal here is more modest: explore the value of experimental systems with real users in the context of PlanetLab itself and the suite of services developed as part of the Content Distribution Network, or CoDeeN, project built on top of it by the Network Systems Group at Princeton University (codeen.cs.princeton.edu). We've organized our discussion around three main themes:

- Using a system exposes implicit assumptions that lead to new research opportunities;
- Real systems often balance competing objectives rather than optimize along a single dimension; and
- Simple yet robust systems that scale "well enough" are better than more complicated systems that require fragile coordination mechanisms.

One of the most important lessons of PlanetLab is that making a system real and letting users interact with it reveals the next set of research opportunities that were not obvious at the outset. Supporting a continuously running system involves more work than running a prototype just long enough to produce a performance number, but the cost is worth it from a research perspective. Our history over the past five years building and measuring a content distribution network illustrates this point. We began by trying to design redirection strategies that would scale under load yet produce response times comparable to the best-known designs. 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 [9].

It was at this point in 2002 that PlanetLab became operational, deploying it in a realistic setting in hopes of better understanding the efficacy of the algorithms. However, within days of deploying the first version, we learned an important lesson: unanticipated traffic (such as spam) compromises the security of the system, making it unusable. So we augmented CoDeeN with new mechanisms that took this lesson and others into account [10]; our own subsequent experience led to improvements in the system [4].

Within weeks of deploying the new system, we discovered that performance was suffering due to failures within the Domain Name System (DNS). Based on additional observations, we determined the root cause to be unexpected failure of the client-side DNS servers, not the server-side DNS infrastructure that is the focus of most DNS research. In response, we demonstrated how the ideas in CoDeeN—designed to make Web content more available—could be adapted to also make DNS resolution more robust. This experience resulted in deployment of a companion system (called CoDNS [5]) on PlanetLab.

By examining CoDeeN use over time, we next came to realize that users benefit not only from the system's caching of files but also from its selection of network paths. The caching benefits were provided only for smaller files (such as typical text-based Web pages), but large files (such as videos and software releases) were still being relayed (though not cached). However, by introducing a new set of mechanisms on top of CoDeeN, we were able 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 other researchers to solve this problem do not work well in practice and that it is only through a thorough evaluation of various engineering trade-offs that we were able to design a system (called CoBlitz) with robust performance under a range of conditions [2]. However, we had to successively refine the system as we improved our understanding of large file usage, server behavior, and network congestion.

In addition to discovering and solving the next set of problems, these running systems were also a boon with respect to new data. For example, we discovered that many assumptions about DNS name stability and intelligent DNS redirection were wrong and produced data that can be used by other researchers [7]. We also collected data by instrumenting CoDeeN

[1] For a partial bibliography of articles written by network systems researchers see www.planet-lab.org.

The research community's reward structure for experimental systems research IS BIASED AGAINST ARCHITECTURAL WORK.

and observed orders of magnitude more Internet routing failures than have been observed through other observation platforms, resulting in a more accurate model of Internet failure behavior [12]. 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 initiation or ISP connectivity [11]. This research helped develop new heuristics to automatically identify and fix these problems in many scenarios and improve the accuracy of network mapping and modeling. We are currently adapting CoDeeN's network failure monitoring to be a continuously running service that reports Internet failures in real time, helping facilitate more network failure research and allowing 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 the original system, as they were in the noise relative to the other factors that actually influence an Internet service. This story likely sounds familiar to developers throughout 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 academic researchers better positioned to frame problems and conduct controlled experiments; one could even argue that much of what they learn through deployment studies is already known by their colleagues in industry. However, our experience suggests two counterarguments:

Tendency to be general-purpose. Academic researchers are much more likely to pursue systems that do not (currently) have commercial value, tending to be enabling and general-purpose (like PlanetLab) rather than address an immediate commercial customer need; and

Access to generally unavailable data. Deploying and instrumenting real systems give researchers access to data not otherwise available; unfettered access to this data—and the ability to add instrumentation as needed—spawns follow-on research.

Think Architecturally

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

Provide a global platform. The platform would have to support both short-term experiments and long-running services associated with a client workload;

Be available immediately. It would have to be available immediately, even though no one knew for sure what "it" would be. This meant we had to evolve PlanetLab incrementally;

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

Minimize centralized components. Sustaining growth would have to depend on support for autonomy and decentralized control; and

Scale for users. It would have to scale to support many users with minimal resources.

PlanetLab supports the required usage model through distributed virtualization, with each service running within a slice of PlanetLab's global resources. Multiple slices run concurrently on PlanetLab, where each slice acts as a networkwide container isolating services from one another.

To address the requirement of being available immediately, PlanetLab adopted an organizing principle called "unbundled management," that is, 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 follows three principles:

- The system should evolve more easily than standard server operating systems;
- Third-party developers should be able to build alternative services, enabling a software bazaar, rather than rely on a single development team

The strategy with PlanetLab has been to architect the system so THE COMMUNITY CAN ADD ENHANCEMENTS (through unbundled management and federation) and, in turn, define a minimal set of interfaces.

with limited resources and creativity; and
- Control over PlanetLab resources and, ultimately, evolution should be decentralized.

Beyond these principles, what made PlanetLab a design challenge was having to resolve conflicts among the various requirements outlined earlier. We do not cover the details of how we resolved them here (see [6]) but highlight the architectural features we introduced to address them.

The first conflict-induced issue was how to minimize centralized components while maintaining the necessary trust assumptions among users and hosting sites. 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 trusted entities are able to peer with one another (using a minimal interface) to support a federation of systems.

The second issue was isolating slices from one another while allowing some slices to manage other slices for the sake of unbundled management. PlanetLab includes a mechanism (called "Proper") that grants slices narrowly defined privileged operations, selectively "poking holes" in the isolation mechanism.

The third issue was balancing the need for slices to acquire the resources they need while coping with scarce resources. PlanetLab's architecture thus includes two features:

Decoupling slice creation from resource acquisition. Slices hold resources only when required by a service or experiment (rather than for the lifetime of the slice), and slices can use alternative resource allocators to acquire the resources they need; and

Depending on "fair share" resource allocation as the default mechanism, augmented with "recovery" mechanisms that deal with potential resource thrashing. One such mechanism kills the slice with the largest physical memory use on a given node when that node's swap space is 90% utilized. It has given users an incentive to be careful about memory consumption, nearly eliminating memory as a bottleneck resource.

The research community's 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 synthesizing 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 user community then comes to depend on these subservices, which now must be maintained, resulting in an ongoing funding challenge for the research community.

Keep it Simple

In many instances of system development, we adopted a simple approach as an initial stepping stone, only to find it is also more desirable long-term than other approaches. In others, we intentionally chose a simple approach to gain robustness. While most testbeds expect only one user at a time, PlanetLab was designed to allow multiple users to share its resources and support long-running distributed services. For simplicity, Linux was chosen as the per-node operating system, expecting to revisit the decision later when virtual machines and other operating systems would be more of an issue. In retrospect, choosing a single operating system reduced management overhead while allowing us to develop the kernel customization needed to scale with PlanetLab's growth. Nodes now regularly run 50 to 100 simultaneous experiments, enabled in part by the use of the VServer mechanism, which virtualizes the filesystem and userspace while running only one OS kernel [8].

PlanetLab's popularity motivated the case for more control over resource allocation, with several competing proposals. We chose Sirius [6], a system for handling resource reservations, to allow any slice on PlanetLab to request a one-hour reservation of guar-

anteed CPU and link resources across PlanetLab. Conventional wisdom suggests that 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 each has its own adaptation mechanism to handle a range of node capacities, including heavily subscribed nodes. The short-term experiments sometimes use it but generally don't run often enough to oversubscribe Sirius. As PlanetLab grows, we may look to enhance Sirius, since it provides a boost across all PlanetLab nodes, with most of its users running on only a subset of nodes.

Simplicity has also guided the longest-running monitoring service on PlanetLab, called CoMon [3], 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 while reducing monitoring CPU utilization on the nodes. CoMon generally consumes about 0.3% of each node's CPU; even monitoring 750 nodes every five minutes, it consumes less than 1Mbps in aggregate. While traffic consumption scales linearly with the number of nodes at the centralized collector, monitoring bandwidth stays constant at each node—important when monitoring nodes with network problems. This design also makes it much simpler to collect certain statistics (such as median values and top values).

CoBlitz, our large file system, is one of the more interesting examples of simplicity in design, with most design decisions running 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 content distribution networks have on the order of 1,000 points of presence. While running directly on end-user machines may sound appealing, security considerations prevent HTTP-compatible systems from taking this approach. BitTorrent, in contrast, can have the server provide content hashes, mitigating the risk, but CoBlitz is designed to run using standard Web servers and unmodified Web clients. This requirement has also driven us to use regular transmission control protocol (TCP) connections, contrary to the finer granularity that the user datagram protocol provides for the current generation of distributed hash tables. However, the incidental benefit of TCP is that it also has fewer problems with stateful firewalls and intrusion-detection systems. Despite these seemingly "unadvanced" design decisions, CoBlitz remains one of the busiest services on PlanetLab and the only large-file service that has stayed in operation since its inception. We attribute at least part of this longevity to the observation that simplicity has made the system more robust and easier to debug and 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 the community can add enhancements (through unbundled management and federation) and, in turn, define a minimal set of interfaces. We cannot claim to have gotten these interfaces completely right yet, but the strategy is a central design principle of the effort.

Conclusion

Classical science equates experimentation with running controlled (lab) experiments that in computer science are often designed to evaluate implementation and engineering choices. However, computer science also benefits from deployment studies (field trials) that involve building and running prototypes subjected to real use. Building something and watching it run thus helps us identify implicit assumptions, the need for different kinds of functionality, surprising behavior, and unexpected limitations. In this sense, working with experimental systems is like constructing a building; engineering principles tell us whether the design is sound, but we need to build it and use it to decide how well it serves its purpose.

We have illustrated how such studies have been beneficial for PlanetLab and the suite of network services that run on top of it. Deployment studies are valuable in exposing new research opportunities, forcing designers to think "architecturally" across the complete system rather than focus on a single dimension, and showing that real systems benefit from simple design, effectively discouraging the temptation to introduce complexity for complexity's sake. C

References

1. Anderson, A., Peterson, L., Shenker, S., and Turner, J. Overcoming the Internet impasse through virtualization. *IEEE Computer 38,* 4 (Apr. 2005), 34–41.
2. Park, K. and Pai, V. Scale and performance in the CoBlitz large-file distribution service. In *Proceedings of the Third Symposium on Networked Systems Design and Implementation* (San Jose, CA, May 8–10). USENIX Association, Berkeley, CA, 2006, 29–44.
3. Park, K. and Pai, V. CoMon: A mostly scalable monitoring system for PlanetLab. *ACM SIGOPS Operating Systems Review 40,* 1 (Jan. 2006), 65–74.
4. Park, K., Pai, V., Lee, K.-W., and Calo, S. Securing Web service by automatic robot detection. In *Proceedings of the 2006 Usenix Annual Technical Conference* (Boston, MA, May 30–June 3). USENIX Association, Berkeley, CA, 2006.
5. Park, K., Pai, V., Peterson, L., and Wang, Z. CoDNS: Improving DNS performance and reliability via cooperative lookups. In *Proceedings of the Sixth Symposium on Operating System Design and Implementation* (San Francisco, Dec. 6–8). USENIX Association, Berkeley, CA, 2004, 199–214.
6. Peterson, L., Bavier, A., Fiuczynski, M., and Muir, S. Experiences building PlanetLab. In *Proceedings of the Seventh Symposium on Operating System Design and Implementation* (Seattle, WA, Nov. 6–8). USENIX Association, Berkeley, CA, 2006, 351–366.
7. Poole, L. and Pai, V. ConfiDNS: Leveraging scale and history to improve DNS security. In *Proceedings of the Third USENIX Workshop on Real, Large Distributed Systems* (Seattle, WA, Nov. 5). USENIX Association, Berkeley, CA, 2006.
8. Soltesz, S., Potzl, H., Fiuczynski, M., Bavier, A., and Peterson, L. Container-based operating system virtualization: A scalable, high-performance alternative to hypervisors. In *Proceedings of EuroSys 2007* (Lisbon, Portugal, Mar. 21–23, 2007).
9. Wang, L., Pai, V., and Peterson, L. The effectiveness of request redirection on CDN robustness. In *Proceedings of the Fifth Symposium on Operating System Design and Implementation* (Boston, Dec. 9–11). USENIX Association, Berkeley, CA, 2002, 345–360.
10. Wang, L., Park, K., Pang, R., Pai, V., and Peterson, L. Reliability and security in the CoDeeN content distribution network. In *Proceedings of the 2004 USENIX Annual Technical Conference* (Boston, MA, June 27–July 2). USENIX Association, Berkeley, CA, 2004, 171–184.
11. Zhang, M., Ruan, Y., Pai, V., and Rexford, J. How DNS misnaming distorts Internet topology mapping. In *Proceedings of the 2006 USENIX Annual Technical Conference* (Boston, June). USENIX Association, Berkeley, CA, 2006, 369–374.
12. Zhang, M., Zhang, C., Pai, V., Peterson, L., and Wang, R. PlanetSeer: Internet path failure monitoring and characterization in wide-area services. In *Proceedings of the Sixth Symposium on Operating System Design and Implementation* (San Francisco, Dec. 6–8). USENIX Association, Berkeley, CA, 2004, 167–182.

Larry Peterson (llp@cs.princeton.edu) is the Robert E. Kahn Professor of Computer Science at Princeton University, Princeton, NJ, and director of the Princeton-hosted PlanetLab Consortium.
Vivek S. Pai (vivek@cs.princeton.edu) is an associate professor in the Department of Computer Science at Princeton University, Princeton, NJ.

This work was funded in part by National Science Foundation grants CNS-0520053, CNS-0454278, and CNS-0335214 and by Defense Advanced Research Projects Agency contract N66001-05-8902.