Learning from PlanetLab

Thomas Anderson
University of Washington

Timothy Roscoe
Intel Research Berkeley

Abstract

PlanetLab has been an enormously successful testbed for networking and distributed systems research, and it is likely to have a significant influence on future systems. In this paper, we examine PlanetLab's success, and caution against an uncritical acceptance of the factors that led to it. We discuss nine design decisions that were essential to PlanetLab's initial success and yet in our view should be revisited in order to better position PlanetLab for its future growth.

Introduction

PlanetLab was designed and constructed as a global platform for supporting distributed systems and network research [23]. Although not the first project to attempt this (earlier examples include CAIRN [13], Active Networks [31, 28], the MBone [15], NIMI [21], Access [29], and RON [3]), PlanetLab has been by far the most successful. When PlanetLab was started almost five years ago, distributed systems and network researchers lacked any meaningful way of deploying their code in the wide area. Today, PlanetLab spans over three hundred sites in thirty countries, hosting over five hundred research projects [22]. More importantly, it has become part of the standard methodology of distributed systems research to evaluate research ideas on PlanetLab prior to publication.

However, in our view PlanetLab has not yet reached its full potential. PlanetLab is widely used for experiments, but that was only half the vision – the other was to foster a new generation of distributed services. It is true that PlanetLab-hosted services communicate with over a million hosts per day, transferring over 4 TB of traffic [22] – services that researchers could not have contemplated building without PlanetLab. More telling, in our view, is that *none* of those services have proven sufficiently popular to drive increased deployment of PlanetLab nodes.

With few exceptions, no-one has joined PlanetLab or upgraded their sites in order to gain better access to the services that run on the platform – PlanetLab's growth has been driven almost entirely by researchers wanting to run their own experiments on the basic PlanetLab infrastructure. In other words, the hoped-for "ecosystem" of services and tools built by and for the research community has not appeared.

The few exceptions have been organizations deploying their own "private" PlanetLab, disconnected from the public PlanetLab, as a way of better managing their own intranets. These "MyPLC" deployments are encouraging, but since they do not peer with the "public" PlanetLab, they do not contribute directly to the platform as a whole. The result can be seen in the rate of new public PlanetLab deployments, currently 15% per year, down from over 100% per year just two years ago.

It is our thesis that design decisions that have made PlanetLab successful have also set it up to fall short over the long term – in other words, "what makes you strong kills you" [8].

We proceed by listing nine fundamental design decisions in PlanetLab, each individually well-intentioned, that we argue have collectively limited PlanetLab's potential to foster the development of a robust set of services running on PlanetLab. Some of these observations have been made before, but we feel a synthesis is timely. Given that the community is in the midst of designing a successor to PlanetLab, our goal in writing this paper is not to point fingers – after all, the authors are among PlanetLab's designers, and as we argue below, many of these decisions were instrumental in PlanetLab's success to date. Rather, our goal is to help define the agenda for the next generation PlanetLab. By not blindly following the recipe for PlanetLab's past success, we argue we can build a system that can achieve PlanetLab's enormous potential as a global platform for distributed and network services.

1 Centralize trust

A key design decision taken early in PlanetLab's lifetime was to centralize trust: PlanetLab Central (PLC) was designated as a trusted intermediary between node owners and node users. In practice, PLC is in control of everything having to do with the public PlanetLab: what operating system is run on each node, whether a particular user or service is allowed to run on a specific node, which services are given special permissions, what resources are allocated to each service, and so forth. This is despite the fact that nominally the resources in PlanetLab are "owned" by the hosting organization. Control is universally delegated to PLC, which in turn operates PlanetLab.

On the positive side, in a small scale system, centralized trust builds robustness for both site administrators and researchers using the system. PlanetLab has been remarkably successful at a key deployment hurdle: to convince site administrators that hosting a PlanetLab node running arbitrary experimental code would not open their networks to attack (trust of remote users has also been a significant hurdle limiting Grid deployments [27]). By controlling the operating system on each node, PLC can ensure that kernel security patches are applied in a timely fashion, reducing the like-

lihood of node compromise. If a research experiment goes awry, site administrators can turn to PlanetLab Central to suspend the slice system-wide, at the first sign of trouble.

Similarly, PLC limited the power of site administrators to customize their systems; for example, site administrators cannot decide what services are provided on the node, which slices are allowed to run, or (except by email request to PLC) how many resources are to be provided to a given slice. Many site administrators, if given the flexibility to do so, would have crafted their own Acceptable Use Policy (AUP) for their nodes; navigating several hundred AUP's would have been quite difficult in practice for experimenters in the early PlanetLab.

While perhaps essential early in PlanetLab's lifetime, we argue that centralized trust is unsustainable over the long term. Clark et al. argue this point: interfaces should be designed to foster competition both above and below the interface, and steps should be taken to avoid inadvertently creating a natural monopoly over the interface [9]. Suppose industry took over operation of the PlanetLab platform, would we be happy if only a single company was allowed to run the system? There is little in the PlanetLab architecture that would prevent a monopoly from occurring. As an analogy, when the Internet's operation was moved from the non-profit to the commercial sector, would we have been happy if there was only room for a single worldwide Internet service provider?

What could we do instead? We believe PlanetLab should be redesigned with flexible, explicit trust. This would mean real competition – node owners get to select who manages their nodes, what runs on it, and how many resources those nodes receive. In the short term, this might result in chaos – different nodes configured in radically different ways, making it more difficult for experiments to span the entire system. But decentralized trust could also yield a diversity of management styles. To take one example, RON has long been managed according to different principles than PlanetLab. Only a few, well chosen experiments were allowed on RON, while PlanetLab was managed to support largest and greatest variety of simultaneous experiments. The result is that packet forwarding services can effectively run on RON, but are not well supported on PlanetLab today [3, 6]. The best way to discover the best management styles is to foster this competition, rather than to suppress it.

2 Centralize resource control

Another key design decision in PlanetLab is that almost all resources are managed centrally by PLC; the owners of individual nodes have only very limited control over their own resources. For example, a node owner may ask PLC (via email) to increase the resource allocation given to a specific slice, but may not decrease it or bar it from running. Note that this is orthogonal to central trust: PLC could choose to delegate its power over resources to the sites owning the individual nodes.

As with trust, the centralization of resource control was important in getting PlanetLab running quickly. Central allocation also allows PlanetLab to provide a uniform view of the system; every slice is able to receive a fixed share of resources on every node in the system. Further, PlanetLab typically operates in a highly resource constrained environment [22]. Central allocation allows PLC to favor slices that serve PlanetLab's end goals, such as those that are widely used or perform essential management services. The notion of centralized control is taken even further in VINI, where the solution to resource scarcity is to give PLC the power to grant a few slices priority over the entire system [6].

However, a perverse side effect of central resource allocation is to reduce the incentive on both site administrators and users to address resource scarcity by provisioning more resources and reducing resource waste, respectively. As just one example, a quarter of all PlanetLab nodes are off-line at any given time, yet the owners of those nodes are free to launch experiments on other nodes that are better maintained. As another example, there is little incentive for PlanetLab users to use less than their fair share of CPU, memory or disk space, regardless of whether that use is productive.

What could we do instead? Some have argued for a PlanetLab-wide virtual currency that would be handed out by PLC for desirable behavior (such as upgrading nodes or keeping their nodes online) and deducted for undesirable behavior (such as triggering security alarms) [17, 4]. While this would help, we believe a central currency is the wrong approach because it requires centralized trust in whoever mints the currency, trust that we would ultimately regret granting. Using real currency instead of virtual currency would avoid issues of trust, but would encourage strategic gaming and might significantly discourage PlanetLab's reason for existence: widespread experimentation and use. Instead, we advocate solutions based on bilateral peering [10, 12], to provide a mild incentive that is nevertheless sufficient to engage people in using and providing resources responsibly. It remains an open research challenge how best to make such an approach effective and easy to configure.

3 Decentralize management

A stated goal of PlanetLab has always been decentralized management [5]: it should be easy for third parties to plug new management services into PlanetLab. A compelling reason for this is to enlist the community in building infrastructure services. By design, PlanetLab provides only the barest management services to users and leaves the remainder to be developed by the community. The idea was to foster innovation and competition among these services, rather than forcing a specific solution.

With very few exceptions, this hasn't worked out in practice – a point eloquently made by Cappos and Hartman [7]. Almost all non-PLC management services were developed in Princeton, co-located with PLC, or at insti-

tutions with close ties to Princeton. Several third-party management services that did appear have been abandoned (e.g. [26, 32, 30]). Users are left with the worst of both worlds – few services and little progress towards improving matters.

As Cappos and Hartman [7] point out, there can be a lack of incentive for researchers to develop long-running services. It is hard work to develop a service, work that does not always pay out in terms of research reputation. However, in our view, incentives are not the whole story, since some researchers have published successfully based on their experience in "real" service deployment and maintenance [25, 20, 22].

We claim instead that the underlying reason for the lack of deployment and adoption, particularly for infrastructure services, is that the PlanetLab trust model discourages innovation and competition in management applications. Most new management services either require some type of privileged operation from PLC, or PLC to make changes in the core PlanetLab software, or both. As we describe below, the PLC interfaces were not designed to be general, but rather to evolve as needed. Thus, in practice, deployment of a proposed new service depends crucially on PLC's willingness and ability to commit to supporting the new service.

This provides an effective disincentive to innovation even if the support is almost always (eventually) provided. A researcher considering whether to put the effort into building a production quality infrastructure service must face the possibility that PLC won't view it as an important component of the wider PlanetLab, no matter how attractive or useful it might be to individual site administrators or to individual researchers. This cuts both ways. Naturally PLC can't adopt an infrastructure service in advance (or put significant development effort into supporting its requirements), without knowing how useful, robust, and reliable it will be.

In retrospect, there is a clear contradiction between centralized trust and resource control on the one hand, and decentralized management on the other. The result is glacially slow progress towards building services that would round out the platform.

What could we do instead? We observe that it is much easier to improve an existing system than it is to create one from scratch. With the benefit of hindsight, we believe it is now possible to identify a key set of management services, such as a debugger, global file system, process manager, and so forth. We believe initial versions of these services should be built and supported, not to dictate a single solution, but rather to provide a (low) benchmark that the PlanetLab community could work over time to improve. Only by removing all possible barriers to researchers deploying infrastructure services can we expect the ecosystem to flourish.

4 Treat bandwidth as free

PlanetLab by design does not charge users for bandwidth. Making resources free fosters use. While it might be difficult for a novel experimental service to justify in advance the bandwidth charges it might incur system-wide, over time it might be able to demonstrate its value to hosting institutions.

However, bandwidth can be a significant cost of operating a PlanetLab node. A single experiment blasting at the full rate of a fast Ethernet could cost a local site over $100,000/year at current market rates of $0.20/GB transferred [2]. In some locations bandwidth can be obtained more cheaply, but in others it is much more expensive – the University of Canterbury in New Zealand, for example, was one of the original PlanetLab sites, but soon afterwards had to shut down their nodes due to excessive bandwidth charges. In practice, most operational PlanetLab sites place strict limits on the maximum rate a node can transfer; the average PlanetLab node in practice incurs bandwidth costs of less than $1,000/year.

The lack of accurate cost accounting has two pernicious effects. The first is to encourage the development of services that use bandwidth in ways that can't possibly be supported over the long term. For example, the heaviest bandwidth consumer on PlanetLab is a content distribution service that advertises that a key benefit is to shift bandwidth costs from the content provider to PlanetLab [11]. The second is to discourage the development of services that use bandwidth in cost-effective ways. For example, it would be impossible on today's PlanetLab to provide a performance-competitive global file service to replace NFS, given current per-node bandwidth limits. Who would volunteer to use an experimental file system that could serve data at a maximum of 10Mbps, the maximum bandwidth allowed at many PlanetLab sites? Yet with effective caching, most of the bytes served by the file system are likely to be to local users and therefore would not even incur wide area bandwidth costs.

What could we do instead? We believe that it is necessary to reflect true bandwidth costs to services, either implicitly through bandwidth trading between peers, or explicitly through cash transfers. Equally importantly, a fine-grained charging model is needed, to reflect that bandwidth to local addresses is typically free and to reflect that some sites are charged only for peak instead of average usage. Even so, research challenges remain. The level of indirection that makes PlanetLab so valuable can foil the accounting that is needed to make PlanetLab realistic. For example, a PlanetLab service that compresses client data before sending it out might appear to incur high bandwidth charges because the data seems to come from PlanetLab rather than directly from the client, even though the compression service is *reducing* overall bandwidth usage across the site.

5 Provide only best-effort service

PlanetLab is designed to provide only best effort service to each slice. PlanetLab is implemented on Linux, and Linux offers only limited support for real-time guarantees. PlanetLab goes beyond this, however, in placing no practical limit on how many slices can be run on a particular node (the number of total slices in the system is somewhat arbitrarily capped at about 4000, ten per site). Because every node is open to all comers, it is easy for experiments to be tested at scale across the entire system.

The downside of this approach, however, is that certain types of experimental services can crowd out other types. We noted earlier that PlanetLab is often resource constrained. If a slice uses a sufficiently large amount of disk space, that can prevent others from running. If several CPU intensive slices share a node, scheduling delay can crowd out any services that need real-time or predictable response time. Recent work has proposed addressing this last problem by granting some slices priority over the entire system [6]. However, this solution requires central trust, is not scalable to PlanetLab-like workloads, and abandons the benefits of best effort service.

What could we do instead? A simple fix would be to cluster experiments by their resource usage profile. The average PlanetLab site has at least two nodes. If a slice runs rarely and uses few resources, it can get to run on one of the nodes; if it uses lots of resources, it is automatically restricted to running only on the node reserved for heavy users (note that full process migration is not needed; most PlanetLab slices are designed to gracefully handle being killed and restarted). By contrast, today PlanetLab slices typically configure themselves to run on the most lightly loaded node; this has the perverse effect of uniformly spreading out heavy users. One nice consequence of this design would be that novice users would always find PlanetLab unloaded, thereby encouraging more use. A longer term research challenge is to redesign the Linux scheduler to better support the diversity of resource demands posed by PlanetLab-like workloads.

6 Make Linux the execution environment

PlanetLab distinguishes between the *execution environment* in which services run and the *API* for explicitly talking to PlanetLab infrastructure [24].

An early design decision in PlanetLab was to provide experimenters with the standard Linux execution environment on each node. Because some slices need privileged services, such as the ability to send an arbitrary packet, a modified version of Linux is now run that provides each slice the illusion that it is running as root on a dedicated node, by catching and translating kernel system calls [18].

Providing a nearly unmodified Linux API has been crucial to PlanetLab's success, by providing a familiar programming environment that is minimally constraining on the behavior of each slice. However, the decision to run on top of Linux has come at a cost. Even ignoring the security vulnerabilities of Linux, it provides very weak isolation between experiments. For example, physical memory is shared, so that one thrashing slice can cause performance problems for all other slices running on the same node. Although virtual machines have been proposed as a solution to the PlanetLab resource isolation problem, the typical PlanetLab node runs many tens of slices at a given time. It is infeasible with today's virtual machine implementations to run more than a few PlanetLab slices, each in their own Linux instance, at a time. While more concurrent slices can run if they are programmed directly on the bare virtual machine, this would impose a high cost in terms of programmability.

At issue here is not necessarily the simple resource overhead of the Linux kernel. While exceptionally large by 1980's standards, the kernel's size and overhead is dwarfed by, say, a Java virtual machine running over it. The issue instead is the constraint on resource policy imposed by the design of the Linux API. Since users aren't charged for the resources they consume, there is no incentive or visibility for them to use resources more wisely.

What could we do instead? We advocate going back to the future. When virtual machines were first developed, IBM built an extremely simple and highly popular single user operating system called CMS [14] that ran only in a virtual machine, and not standalone. CMS could be lightweight because it did not need to do sophisticated resource management; its sole role was to provide a convenient programming interface. We believe something similar is needed today for PlanetLab.

To put a more radical spin on things, there seems little evidence that POSIX is a suitable API for planetary-scale services, any more than it is appropriate for mobile or wireless devices like phones or sensor nodes. Surely as systems researchers we can do better than this, if provided with the ability to deploy service-specific kernels (albeit in a virtual machine)? Now that PlanetLab is the norm for evaluating research distributed systems, it seems counterproductive to require everyone to keep using the same old interfaces, which were never designed for the task at hand.

7 Don't Provide Distributed OS Services

In many respects, PlanetLab is a distributed operating system, but it doesn't act like one: it provides a distributed process abstraction (the slice), but little else. Above we discussed the lack of management services, such as a global file system. Here we discuss kernel level distributed operating system services: security and isolation.

While PlanetLab provides weak isolation between slices at the node level, it provides no isolation at the level of aggregate resources across the entire system. While designing such a distributed bounding box is a research challenge, it is an essential one to prevent PlanetLab from being mis-

used to launch distributed denial of service attacks. Similarly, PlanetLab has only a single type of user; all experiments are equally powerful, even those written by novices. It is equally important to be able to prevent inadvertent distributed denial of service attacks.

What could we do instead? We feel two aspects of the PlanetLab's current design have received insufficient attention to date. The first is resource control "in the large" - it should be possible to talk about the resource usage of an entire slice and make statements about the limits of resource usage of that slice. Such statements might be complex, a canonical example being constraining the volume of traffic to any single destination address from the slice as a whole.

The second, related, aspect is per-slice control of packet transmission and reception on a node: to implement features like a distributed bounding box, the current "special case" of restrictions on raw socket access should be generalized and become the way all slices talk to the network – in other words, enforceable policies describing traffic rates, characteristics, and formats should be the norm for all slices.

8 Evolve the API

The PlanetLab API was designed to evolve: get it running quickly, and learn as we go [22]. While appropriate in the early stages of PlanetLab's life cycle, there is a need now for a thorough rethinking to ensure that the API is sufficiently general to support any legitimate service. To take one example, it should be straightforward to build a debugger for a PlanetLab slice, but it is not given the current API. The most obvious way to support a debugger would be to be able to suspend a sliver, and then examine the sliver's state. However, in today's API, these are privileged operations that only PlanetLab operations staff can perform.

As another example, in an earlier section we described a resource allocation service that migrated slices based on their pattern of resource usage. While there is a sensor interface to export this type of information to a monitoring slice, it is not designed to be general purpose. For example, the sensor interface aggregates data at a time interval chosen by the system designer (today, a second). Thus a slice consuming 100 ms of sequential CPU time is equivalent to another slice consuming CPU time in 100 1 ms intervals, despite the fact that these two profiles are very different in terms of their impact on real-time delivery constraints.

A related issue is that the PlanetLab API is not a standard [22]. Although the API is documented, the documentation has frequently been out of date, is not in machine-readable form suitable for stub compilers, and is maintained by hand rather than being generated from a specification. Furthermore, there is no community approval process for proposed changes; the API is whatever PLC defines it to be at a given moment. On the positive side, this makes the system highly adaptable; problems can be fixed and functionality added (by PLC) as soon as needed. However, on the downside, this means that developers have little guarantee of stability or sustained backward compatibility. Several infrastructure services have been abandoned over the past few years due to changes in the API (for example, [26, 32, 1]), not only wasting effort, but also making it less likely for anyone else to step in to adopt the service. Even successful infrastructure services have reported on the difficulties of working with PlanetLab's changing API [7].

What could we do instead? Despite the overhead inherent in any standardization process, we believe it is now time that the ownership of the PlanetLab API be turned over to a standardization process run by the PlanetLab developer community. The goal would be to develop an API that is general purpose enough to be stable over the long run.

9 Focus on the machine room

Finally, we note that PlanetLab by design targets machine rooms. The recommended minimum PlanetLab node configuration has become a hefty server: at the time of this writing, a 3GHz CPU, 4GB of DRAM and 300GB of disk. Over time, the minimum configuration has rapidly increased in power, simply to keep up with the growth in the number of experiments running on PlanetLab.

However, the future of distributed systems is not servers, but ubiquitous low power clients such as cell phones and PDAs. If PlanetLab is to be relevant in the long term, it needs to extend its slice abstraction to be able to run everywhere.

What could we do instead? Development and widespread deployment of services like Oasis [19] and OCALA [16] could enable PlanetLab to run everywhere. Of course, this is itself predicated on providing more fully decentralized trust, incentives, heterogeneous execution environments, and scalable resource allocation (allowing every PlanetLab experimenter access to every PDA is clearly a non-starter). Nevertheless, this seems to us the most viable strategy for expanding PlanetLab's reach out from the research community to more ubiquitous deployment across more diverse networks.

Summary

PlanetLab has been enormously successful, but it could be even more so. Universal access to programmable platforms everywhere in the world is our future. We describe nine decisions that have been crucial to PlanetLab's success but which we argue should be rethought now that PlanetLab is successful. Those familiar with GENI will observe that many of the ideas in this paper have informed our design for GENI. Our goal in writing this paper is to encourage research to understand how to keep the benefits PlanetLab provides, while repositioning it for its future success.

Acknowledgements

The views in this paper are our own, but they have been inspired and enriched by numerous conversations with fellow researchers. In particular, we are indebted to our colleagues in the GENI Distributed Services Working Group for the initial discussion that led to this paper.

References

[1] R. Adams. PlanetLab Slice Tools. `http://jabber.services.planet-lab.org/php/software/tools.ph`, March 2005.

[2] Amazon. Amazon Simple Storage Service (Amazon S3). `http://www.amazon.com/s3`.

[3] D. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris. Resilient Overlay Networks. In *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles*, 2001.

[4] A. AuYoung, B. Chun, A. Snoeren, and A. Vahdat. Resource Allocation in Federated Distributed Computing Infrastructures. In *Proceedings of the First Workshop on Operating System and Architectural Support for the On Demand IT Infrastructure*, Oct. 2004.

[5] A. Bavier, M. Bowman, B. Chun, D. Culler, S. Karlin, L. Peterson, T. Roscoe, and M. Wawrzoniak. Operating Systems Support for Planetary-Scale Network Services. In *Proceedings of the 1st Symposium on Networked Systems Design and Implementation (NSDI'04), San Francisco, CA, USA*, March 2004.

[6] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford. In VINI Veritas: Realistic and Controlled Network Experimentation. In *Proceedings of the ACM SIGCOMM 2006*, Sept. 2006.

[7] J. Cappos and J. Hartman. Why It Is Hard to Build a Long-Running Service on PlanetLab. In *Proceedings of Usenix WORLDS*, March 2005.

[8] C. Christensen. *The Innovator's Dilemma*. Harvard University Press, 1997.

[9] D. Clark, J. Wroclawski, K. Sollins, and R. Braden. Tussle in Cyberspace: Defining Tomorrow's Internet. In *Proceedings of the ACM SIGCOMM 2002*, Pittsburgh, PA, Aug. 2002.

[10] B. Cohen. Incentives Build Robustness in Bittorrent. In *Proceedings of the First Workshop on Economics of Peer-to-Peer Systems*, June 2003.

[11] CORAL. The Coral Content Distribution Network. `http://www.coralcdn.org/overview`.

[12] Y. Fu, J. Chase, B. Chun, S. Schwab, and A. Vahdat. SHARP: An Architecture for Secure Resource Peering. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, 2003.

[13] Information Sciences Institute. Collaborative Advanced Internet Research Network (CAIRN). `http://www.isi.edu/div7/CAIRN`.

[14] International Business Machines. *z/VM CMS Application Development Guide*, version 5 release 2 edition, December 2005. Part number SC24-6069-01, `http://publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/download/hcsd0b10.pdf?ACTION=SAVE&DT=20051013155749`.

[15] Internet Engineering Task Force. MBONE Deployment. `http://www.ietf.org/html.charters/mboned-charter.html`.

[16] D. Joseph, J. Kannan, A. Kubota, K. Lakshminarayanan, I. Stoica, and K. Wehrle. OCALA: An Architecture for Supporting Legacy Applications over Overlays. In *Proc. 3rd USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI '06)*, San Jose, CA, May 2006.

[17] K. Lai, L. Rasmusson, E. Adar, S. Sorkin, L. Zhang, and B. Huberman. Tycoon: An Implementation of a Distributed Market-Based Resource Allocation System. Technical Report, HP Labs, 2004.

[18] Linux Vserver Project. `http://linux-vserver.org`.

[19] H. V. Madhyastha, A. Venkataramani, A. Krishnamurthy, and T. Anderson. Oasis: An overlay-aware network stack. *SIGOPS Oper. Syst. Rev.*, 40(1):41–48, 2006.

[20] K. Park and V. S. Pai. Scale and Performance in the CoBlitz Large-File Distribution Service. In *Proceedings of the Third Symposium on Networked Systems Design and Implementation (NSDI 2006)*, San Jose, CA, May 2006.

[21] V. Paxson, A. Adams, and M. Mathis. Experiences with NIMI. In *Proceedings of Passive and Active Measurement 2000*, 2000.

[22] L. Peterson, A. Bavier, M. Fiuczynski, and S. Muir. Experiences Building PlanetLab. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, Nov. 2006.

[23] L. Peterson, D. Culler, T. Anderson, and T. Roscoe. A Blueprint for Introducing Disruptive Technology into the Internet. In *Proc. HotNets-I*, 2002.

[24] L. Peterson and T. Roscoe. The Design Principles of PlanetLab. *ACM Operating Systems Review*, 40(1), January 2006.

[25] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu. OpenDHT: A Public DHT Service and Its Uses. In *Proceedings of the ACM SIGCOMM 2005*, 2005.

[26] T. Roscoe, R. Mortier, P. Jardetzky, and S. Hand. InfoSpect: Using a Logic Language for System Health Monitoring in Distributed Systems. In *Proceedings of the 10th ACM SIGOPS European Workshop*, pages 00–00, Saint-Emilion, France, September 2002.

[27] J. Schopf and B. Nitzberg. Grids: The Top Ten Questions. *Scientific Programming*, 10(2), 2002.

[28] J. Smith and S. Nettles. Active Networking: One View of the Past, Present, and Future. *IEEE Transactions on Systems, Man, and Cybernetics*, 34(1), 2004.

[29] Q. Wang. Access: A Communication and Computation Environment for Wide Area Systems Research. Masters Thesis, University of Washington, 2000.

[30] M. Wawrzoniak, L. Peterson, and T. Roscoe. Sophia: An Information Plane for Networked Systems. In *Proceedings of the 2nd ACM Workshop on Hot Topics in Networks (HotNets-II)*, November 2003.

[31] D. Wetherall. Active Network Vision and Reality: Lessons from a Capsule-Based System. In *Proceedings of the Seventeenth ACM Symposium on Operating Systems Principles*, 1999.

[32] C. Yoshikawa. PlanetLab Map Application. `http://www.ececs.uc.edu/~yoshikco/planetlab/planetlab.htm`, March 2005.