Why it is hard to build a long-running service on PlanetLab

Justin Cappos and John Hartman
Computer Science Department
University of Arizona
Tucson, AZ, 85721
{justin,jhh}@cs.arizona.edu

Abstract

PlanetLab was conceived as both an experimental testbed and a platform for long-running services. It has been quite successful at the former, less so at the latter. In this paper we examine why. The crux of the problem is that there are few incentives for researchers to develop long-running services. Research prototypes fulfill publishing requirements, whereas long-running services do not. Several groups have tried to deploy *research services*, long-running services that are useful, but also novel enough to be published. These services have been generally unsuccessful. In this paper we discuss the difficulties in developing a research service, our experiences in developing a research service called Stork, and offer suggestions on how to increase the incentives for researchers to develop research services.

1 Introduction

Building a long-running service on PlanetLab[12] is difficult. PlanetLab has been used extensively to test and measure experimental research prototypes, but few long-running services are in wide-spread use. At the root of the problem is the "dual use" purpose of PlanetLab as originally conceived. On the one hand, PlanetLab is a research testbed, giving researchers access to numerous geographically-distributed nodes, realistic network behavior, and realistic client workloads. The bulk of PlanetLab activity to date has been of this sort. Most of the services that run on PlanetLab are short-lived *research prototypes*, developed as part of various research projects and funded by research funding agencies. As such, the prototypes exist to support experiments and produce publishable results. They must be sufficiently novel and must function only well enough to run the necessary experiments and collect the necessary results to validate the design.

On the other hand, PlanetLab is supposed to be a platform for deploying long-lived services, connecting researchers who want to produce these services with the users who want to use them. Presumably these services are based on earlier research prototypes, and incorporate novel features that the users find appealing. To date, very few of these long-lived services have been deployed and even fewer have come into wide-spread use.

The reasons for this lie in the lack of incentives for researchers to produce long-lived services. The original PlanetLab paper is notably silent on this subject: PlanetLab's dual purpose is touted as the most distinguishing characteristic of the PlanetLab approach to changing the Internet, but no blueprint is given for how this dual use will come to be. Most of the discussion revolves around how to support it, rather than how to make it a reality.

As a result, there are few long-lived services on PlanetLab. PlanetLab is designed and used by researchers for whom the reward structure provides little incentive to convert research prototypes into long-lived services. Research prototypes suffice for publication; since the reward for developing a new research prototype is quite high, and the reward for converting an existing prototype into a long-lived service is quite low, it is little wonder that PlanetLab is awash in prototypes while suffering a drought of services.

A potential middle ground is the *research service*, a long-lived service with sufficient research content to warrant publication. They are more interesting than simple services, and more available and reliable than research prototypes. Unfortunately, by spanning the gap between simple services and research prototypes, research services must meet the requirements of both worlds (Figure 1). They must have a research component so as to fit into the standard research reward structure, yet must not have any corner cases that make them unreliable. They must be long-lived, yet permit experimentation. There are several reasons why it is extremely difficult to build a successful research service:

- Research services must contain a research component that supports a research hypothesis. They must do something new and interesting, rather than established and mundane, as the reward system values novelty.
- Research services must rely on other research services. A research service need not be novel in all respects, but in those areas where it is not, it is expected to make use of the current state-of-the-art re-

	Research Prototypes	Research Services	Services
Reliability	Unimportant	Very Important	Very Important
Novelty	Very Important	Very Important	Unimportant
Research Interest	High	High	Low
Example Services	Bullet SHARK	Stork Bellagio CoBlitz	Sirius CoMon AppManager

Figure 1: *Characteristics of research services, services, and research prototypes.*

search results. If, for example, recent research has shown that reliable data transfer is best achieved using a particular technique, a research service that incorporates reliable data transfer will be expected to use that technique. That means that research services end up depending on one another. Since a well-known aphorism is to avoid having one's research depend on another research project, this is not a recipe for success.

- Research services are prone to instability. Research prototypes ignore the corner cases for a reason – they are difficult to handle and don't contribute to the research results. This means that a research service tends to have a relatively high bug/failure rate. Interaction between research services makes the problem worse. Each additional service introduces its own set of corner cases, increasing the size of the corners and decreasing overall stability.

For the remainder of the paper we describe the development cycle of a PlanetLab research service called *Stork*. We then discuss the issues and problems that cause research services to enter a downward spiral on PlanetLab, and conclude with suggestions on how to break the cycle by providing incentives for researchers to develop research services rather than research prototypes.

2 Stork

Stork is a PlanetLab research service that installs and maintains software for other services. A key problem facing PlanetLab services is the difficulty in installing software on a large set of nodes and keeping that software updated over a long period of time. Researchers need to quickly and efficiently distribute new package content to huge numbers of nodes, and do so in the face of network and node failures. In such an environment, nodes may miss software updates, however the correct software state must eventually be reached. In addition, software must be installed on the nodes efficiently. Each node runs hundreds of slices, many of which will install the same software. Having hundreds of copies of the same software on a node is not feasible; provisions must be made for sharing copies between slices.

Stork solves this software maintenance problem. Software installed using Stork is securely shared between slices, and a package is not downloaded if another slice on the same node already has it installed. Stork uses efficient transfer mechanisms such as CoBlitz[4] and BitTorrent[5] to transfer files. Stork has security features that allow developers to share a package on a repository without trusting each other or the package repository administrator.

Stork uses the functionality provided by other long-running services to enhance its capabilities. Stork uses Proper[10] to share and protect content, CoDeeN[19] and CoBlitz to transfer files, and AppManager[8] to deploy itself on every node.

We first give an overview of the Stork project throughout its development. We then discuss the specific problems that we encountered during different phases of development and how these problems are instances of the more general problems that plague research services.

2.1 Timeline

We present a timeline for Stork in Figure 2. The timeline illustrates the different phases of the project during development and deployment. Each period begins when the first line of code for a version was written. The resources, implementation and tool features for each version are also discussed.

Throughout the design and development of Stork we performed incremental roll-out and development. Our intention was to gradually build Stork's capabilities along with its user base, relying on user feedback to help define its development.

2.1.1 Plan-apt (Stork predecessor)

Plan-apt is a precursor to Stork that we developed for the purpose of installing and updating packages on PlanetLab nodes. Plan-apt is essentially a simple remote execution program with functionality tailored to package installation. It allows a single host to push package updates to many client slices. Unfortunately, since plan-apt required the user to start a daemon in each slice to be managed, the setup cost was unattractive.

We were frustrated by the inefficiencies of plan-apt with regard to setup, disk usage, and security. We decided to shelve remote execution to instead focus on a tool that securely shares package content between slices on a single node.

2.1.2 Stork (Alpha release)

We developed the first version of Stork to address plan-apt's shortcomings. This version uses apt to fetch packages and resolve dependencies, but provides extra security and disk space savings. It was mostly focused on

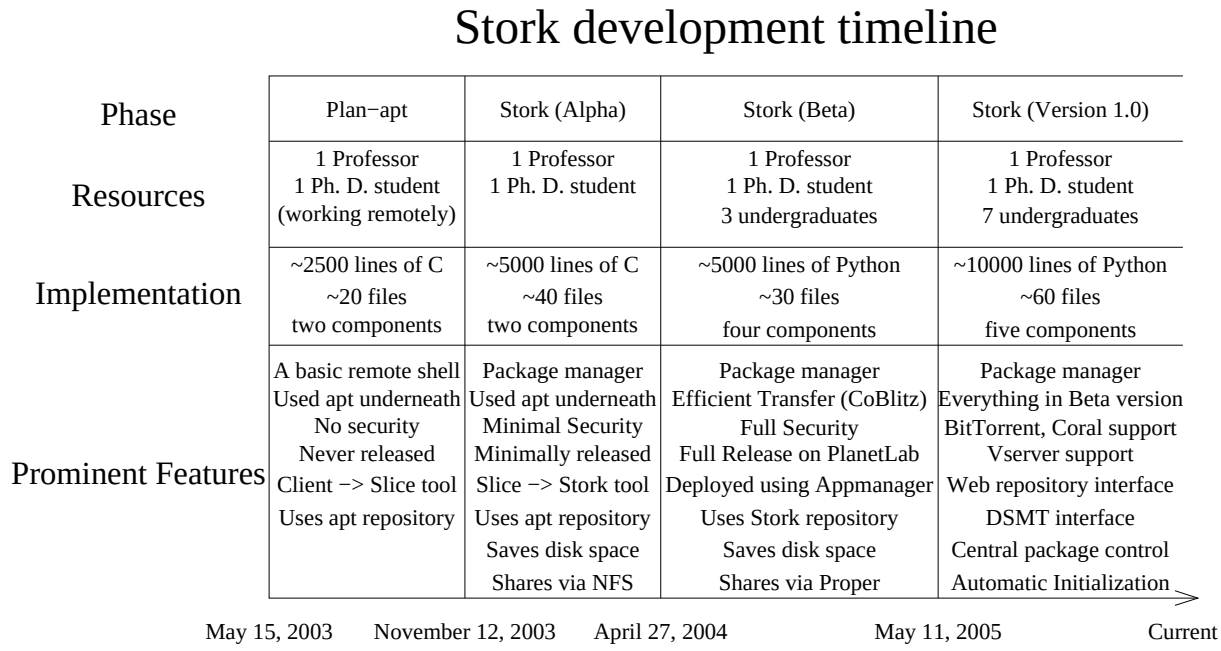

Figure 2: *This timeline shows the evolution of the Stork project. We show the resources used during development, the method of implementation, and the prominent features of each release.*

creating a package manager that efficiently and securely shares content between slices using NFS.

2.1.3 Stork (Beta release)

In the next iteration we rewrote Stork using Python to clean up the code and provide additional functionality missing in the alpha version. We also developed Stork into an independent package management tool no longer reliant on apt. We added support for additional transfer methods and package types. This version of Stork uses packages primarily in the RPM format and resolves dependencies itself. We also use Proper to securely share packages between slices using hard links and file immutable bits. This provides a fast and transparent method to share files instead of NFS.

We divided Stork into four components to handle different tasks. As in the previous version there is a Stork slice on each node as well as a set of installation tools, but the beta version also added repository scripts and a set of tools to authorize package use. Users digitally sign packages and specify to Stork which other users' digital signatures they trust for groups of packages. Stork verifies package signatures before installing a package in a slice.

2.1.4 Stork (Version 1.0)

The latest version of Stork features another re-write and additional functionality. We changed to a more modular design that allows developers to write simple stubs to perform similar actions using different implementations. For example, Stork has a stub interface for network transfers with stubs such as HTTP, CoDeeN, FTP, CoBlitz, and BitTorrent available. A developer could create a new stub for Bullet[9] and then use that stub with Stork without modifying any other code.

2.2 Problems

The problems with different versions of Stork were mainly due to competing interests. Since Stork is a research service, we tried to balance research and usefulness. Where practical, we made use of other research services so as to increase Stork's functionality. For example, Stork downloads content using CoBlitz and BitTorrent, shares files across slices using Proper, and has a novel technique for validating packages. The complexity of providing these features has greatly decreased the usability and stability of Stork as a whole.

When developing the beta version of Stork we decided to include intelligent content transfer. Point-to-point HTTP transfers worked fine for the loads experienced by earlier versions of Stork, but clearly would not scale to larger numbers of users. As a result we added support for other transfer types, including BitTorrent, CoBlitz, and CoDeeN. We allowed the user to choose what transfer type they wanted to use for their transfer, but unfortunately when a transfer type failed, we did not retry with another type. From a research standpoint this was an appropriate simplification because all of the functionality was there. From a service point of view, it caused failures that our users did not appreciate.

Stork also depends on Proper, a research service that enables inter-slice interaction such as file sharing. This dependency has also proven problematic, as getting Stork and Proper to work well together has been difficult. When an error occurs it isn't clear where the problem lies. Stork must depend on Proper because sharing package files between slices is a key motivation for Stork. Sharing files allows Stork to save disk space, avoid unnecessary information downloads, and reduces the memory use of shared programs. However, these features are more beneficial to the PlanetLab infrastructure than individual users; users want a service that always works, rather than one that works most of the time and provides only intangible benefits.

Downloading packages securely is a major feature of Stork. The standard solution is a package repository that is trusted to contain only valid packages. This security model is inappropriate for PlanetLab's unbundled management. Instead, Stork allows users to sign packages digitally, and only install packages with acceptable signatures. Acceptable signatures can be specified on a per-package basis, allowing a user to accept another user's signature for certain packages but not others.

Although these measures greatly increase Stork's security, it makes Stork more complicated for our users. Users must not only sign packages that they upload to our repository, but they must also configure Stork to accept the appropriate signatures. This complexity has been the source of much confusion and frustration by our users.

3 Making Research Services Viable

The requirements for research and stability oppose each other and create a fundamental tension in research service development. This tension helps to create a downward spiral for each research service on PlanetLab and underscores the gap between research and real world practices. In this section we describe the downward spiral, and offer suggestions for escaping it.

3.1 The Downward Spiral of Research Services on PlanetLab

One problem that most services face is they get into a negative cycle that stunts project development and prevents the growth of a strong user base. A service cannot build a user base because it lacks stability and features. The features and stability cannot be provided without a large user base to drive the feature set and do the necessary third-party testing.

Research services on PlanetLab tend to be either very successful in attracting users (e.g. CoDeploy and Coral[7]) or have few users (e.g. Stork, Bellagio[1], and DSMT[6]). There doesn't appear to be any middle ground. Most of the successful research services

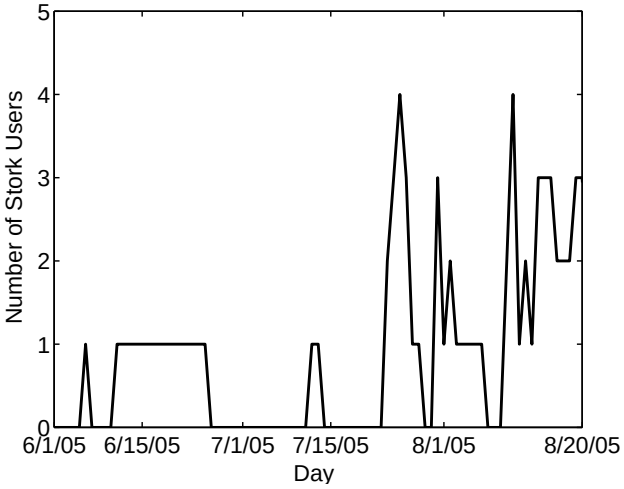

Figure 3: *The number of slices that used Stork on each day from June 1st to August 20th, 2005.*

on PlanetLab are HTTP content distribution networks that provide the same functionality as existing non-PlanetLab services. They do not have dependencies on external research services: either they don't need such services, or the research groups developed the necessary research services themselves. For example, CoBlitz depends on CoDeeN, CoMon, and CoDNS, all developed by the same research group.

The failed research services tend to only be useful for other PlanetLab researchers, and have dependencies on other research services. The former reduces the available user base, while the latter decreases stability. As a research service depends on more and more research services, the size and number of corner cases increases. Eventually, almost the entire operational space is corners, leaving very little of the service functional. This leads users to avoid the service and continues the spiral.

3.2 Escaping the Spiral

We have several suggestions to help improve the quality and usability of research services on PlanetLab.

1. **Fall-back gracefully to independent operation.**
   Although a research service may depend on other research services, it should fall back to more reliable functionality if necessary. For example, Stork uses Proper to share files but will directly download and install them if Proper fails. Stork also falls back to direct HTTP transfers should the more sophisticated download mechanisms fail. In many cases this fall-back does not inconvenience the user, it only decreases the overall efficiency of the system. Since users prefer functionality to efficiency, this seems a reasonable trade-off, at the cost of additional complexity to implement the fall-back mech-

anism.

2. **Build on other research services.**

   The previous suggestion does not mean that research services should never rely on one another. It should just be done in moderation. We believe that inter-dependency is a good thing, as it increases research service functionality and potential user base. To date, we have tried to be active users of any appropriate research service on PlanetLab. Stork uses Proper, CoDeeN, CoBlitz, and AppManager to provide it with improved functionality. We have also investigated using PLuSH[16], DSMT, Coral, Bellagio, PsEPR, and Sirius to various degrees.

3. **There must be a reliable core.**

   While research services are interesting because of the research component, users just want them to work. If a research service provides an operational subset that always works, then users can be assured that there are some functions that are well tested and can be depended upon. In other words, the research service developer should make a reliable service and build the research framework around it.

4. **Incentives are needed for research service creation.**

   Services like AppManager work very well in practice, but are uninteresting from a research standpoint. For that reason few of them exist on PlanetLab, although those that do exist are stable and well-used. The motivation for creating such services isn't clear, but may be similar to that of people who write open source software. Service creators obtain a certain amount of renown within the community but largely donate their effort with the hope that others will also donate useful software.

   If the PlanetLab Consortium were to offer prizes for service creation and deployment (similar to the Ansari X-Prize [20]), it would increase the interest in service creation. Currently there are too many research prototypes and too few services to have a stable base to build upon.

   Perhaps PlanetLab should start its own conferences and/or journals that focus on research services that have real user bases. WORLDS is a step in the right direction, but workshop publications don't have the same weight as conferences and journals. Not only would these publications help other researchers to develop research services, it would give them an incentive to do so. Alternatively, sessions which focus on high-quality experiences papers could be added to existing conferences.

5. **Standardized interfaces are not the solution.**

   The problem with inter-service dependency is not the lack of standardized interfaces. Dealing with different interfaces for different services is not that difficult; adding a new data transfer service to Stork is as simple as writing a few stub routines. The problem is running into corner cases in which the research service does not work correctly. Working around a bug in another service is extremely difficult and time-consuming. Effort should be expended reducing corner cases and documenting those that remain, rather than standardizing interfaces.

4 Conclusion

Throughout this paper we have described the problems that research services face on PlanetLab. The requirements for novelty and interuse cause instability that frustrates users. We have provided suggestions explaining how to build a reliable and stable tool for users without sacrificing the research value of the service. New incentives for research services along with better techniques for building research service would help to develop PlanetLab. With time, PlanetLab may fulfill the dream of having long-running research services running alongside research prototypes.

5 Acknowledgements

We would like to thank Vivek Pai and Steve Muir for suggesting we write this paper and the entire Stork team, especially Jason Hardies, for keeping other projects moving while we worked on this. We would also like to thank our shepherd, Dave Andersen, and the anonymous reviewers for their comments that greatly improved this paper.

References

[1] AuYoung, A., Chun, B., Snoeren, A., Vahdat, A., "Resource allocation in Federated Distributed Computing Infrastructures", In Proceedings of the 1st Workshop on Operating System and Architectural Support for the On-demand IT InfraStructure (2004).

[2] Brett, P., Knauerhase, R., Bowman, M., Adams, R., Nataraj, A., Sedayao, J., Spindel, M., "A Shared Global Event Propagation System to Enable Next Generation Distributed Services", First Workshop on Real, Large Distributed Systems (WORLDS), 2004

[3] Brooks, F., "The Mythical Man Month", 1975

[4] CoBlitz, http://codeen.cs.princeton.edu/coblitz/

[5] Cohen, B., "Incentives Build Robustness in BitTorrent", Workshop on Economics of Peer-to-Peer Systems, 2003

[6] Distributed Service Management Toolkit, http://yum.psepr.org/

[7] Freedman, M., Freudenthal, E., and Mazires, D., "Democratizing Content Publication with Coral", In Proc. 1st USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI '04) San Francisco, CA, March 2004.

[8] Huebsch, R., PlanetLab application manager. http://appmanager.berkeley.intel-research.net/

[9] Kostic, D., Rodriguez, A., Albrecht, J., Vahdat, A., "Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh", in Proceedings of ACM SOSP, 2003.

[10] Muir, S., Peterson, L., Fiuczynski, M., Cappos, J., Hartman, J., “Proper: Privileged Operations in a Virtualised System Environment”, USENIX 2005

[11] Peterson, L., “Dynamic Slice Creation”, PDN-02-005 Draft, 2002.

[12] Peterson, L., Anderson, T., Culler, D., Roscoe, T., “A Blueprint for Introducing Disruptive Technology into the Internet”, PDN-02-001, 2002.

[13] Peterson, L., Roscoe, T., “PlanetLab Phase 1: Transition to an Isolation Kernel”, PDN-02-003, 2002.

[14] PlanetLab Sirius Scheduler, http://snowball.cs.uga.edu/ dkl/pslogin.php

[15] Plkmod. http://www.cs.princeton.edu/ acb/plkmod/

[16] PLuSH, http://sysnet.ucsd.edu/projects/plush/

[17] Stork. http://www.cs.arizona.edu/stork/.

[18] Vservers. http://linux-vserver.org/

[19] Wang, L., Park, K., Pang, R., Pai, V., Peterson, L., “Reliability and Security in the CoDeeN Content Distribution Network”, Proceedings of the USENIX 2004 Annual Technical Conference, 2004

[20] XP. http://www.xprizefoundation.com/