72 Hours to DonutLab: A PlanetLab with No Center

Marc Stiegler, Mark S. Miller[1], Terry Stanley[2]
Mobile and Media Systems Laboratory
HP Laboratories Palo Alto
HPL-2005-5
January 6, 2005*

security, distributed computing, PlanetLab, Agoric Computing

PlanetLab [Peterson02] has been developed as a platform for experimenting with globally distributed computing systems. The DonutLab is a conceptual descendant of PlanetLab, intended to share many of PlanetLab's merits while incorporating a number of significant enhancements for security, ease of use, reliability, and persistence. DonutLab was built using the Promise Pipelining Object-Capability (PPOC) programming language ***E***. The primary goal was not merely to demonstrate a direction of evolution for PlanetLab; rather, it was to demonstrate the power of PPOC for building secure distributed systems. To showcase this power, DonutLab was built over a 3-day weekend, in 72 hours. Here we investigate the successes and failures of the 72-hour effort, the nature of the DonutLab, and the features of PPOC that enable the high-speed construction of such easy to use yet secure systems.

* Internal Accession Date Only
[1]Affiliated with Johns Hopkins University and Hewlett-Packard Laboratories
[2]Cocoon.com Consulting
Approved for External Publication

Abstract

PlanetLab [Peterson02] has been developed as a platform for experimenting with globally distributed computing systems. The DonutLab is a conceptual descendant of PlanetLab, intended to share many of PlanetLab's merits while incorporating a number of significant enhancements for security, ease of use, reliability, and persistence. DonutLab was built using the Promise Pipelining Object-Capability (PPOC) programming language ***E***. The primary goal was not merely to demonstrate a direction of evolution for PlanetLab; rather, it was to demonstrate the power of PPOC for building secure distributed systems. To showcase this power, DonutLab was built over a 3-day weekend, in 72 hours. Here we investigate the successes and failures of the 72-hour effort, the nature of the DonutLab, and the features of PPOC that enable the high-speed construction of such easy to use yet secure systems.

Introduction

The tactical goal of DonutLab was to build a PlanetLab-like distributed system that could be used by researchers to build and test distributed systems, with the following properties that go beyond the current PlanetLab implementation:

- **Full Decentralization**: "PlanetLab Central" is a single point of failure that impacts reliability, security, and scalability. DonutLab has no center.
- **Agoric (market-based) Resource Allocation**: By using agoric resource allocation [Miller88, Stonebraker94], we eliminate the risk that the DonutLab could be used as a weapon for Distributed Denial of Service (DDOS) attacks. PlanetLab currently has over 500 computers with high-bandwidth connections, a most attractive attack vehicle for those individuals interested in building a DDOS arsenal [Adams03, Pai].
- **Persistence**: On DonutLab, what goes down will, in general, come back up. Persistence can be particularly difficult in the presence of strong security guarantees: the time of restart can be fraught with significant vulnerabilities. DonutLab uses the persistence system in ***E*** that was designed specifically to enable revival in the presence of complex mutually suspicious trust relationships.
- **Secure Cooperation**: DonutLab servers can be run inside the firewall with minimal risk to any of the assets protected by the firewall. Furthermore, the compromise of any one object in the DonutLab cannot cascade into a large, possibly disastrous, system breach.
- **Ease of Use**: DonutLab, by using an authorization-based paradigm (object-capabilities [Dennis66]), avoids the pitfalls inherent in ID-authentication-based approaches to security. DonutLab has no passwords, certificates, or other obstacles to human action that are usually considered a necessary tradeoff to achieve secure operations.
- **Deadlock-free operation** that maximizes performance in the face of latency: As computers become more powerful, and bandwidth increases, the limiting factor increasingly becomes latency. Promise Pipelining minimizes the impact of latency while guaranteeing that deadlock cannot occur (*datalock*, a loss-of-liveness bug similar in some ways to deadlock, is still possible, but it occurs less often and more deterministically).

DonutLab Structure

The basic DonutLab includes the following elements:

- **Mints**: These are the services that issue currency.
- **SliverServers**: In PlanetLab, the researcher acquires a "sliver" on each machine. This sliver is a virtual machine in which the researcher can run any code he desires. The SliverServer fulfills the same function. The SliverServer charges for clock time on the machine and for messages sent. Clock time, not CPU time, is the basis for charging simply because this was easy to do in 72 hours.
- **Kiosks**: These advertising services enable programs to locate other services in the Donut. The two types of services currently listed are kiosks themselves and SliverServers. Asking the kiosk for a list of servers is free, while posting an advertisement for a service requires a nominal fee. Service providers can supply more money for a better location in the list of servers delivered to potential buyers.
- **Doughbot Sample Application**: The Doughbot is a DDOS attack weapon

designed to destroy the DonutLab. It creates slivers on all the SliverServers, and uses them to attack the kiosks. The goal is to prevent any other applications from using the Donut. Of course, given the Donut's agoric resource allocation, this DDOS attack achieves a rather different result than that intended by the attackers, as discussed later.

A typical operating scenario would involve the following steps:

- The researcher receives 2 secure references, one to an account on a mint, the other to a kiosk. This pair of references, typically embodied as a pair of files, can be sent via PGP mail or any other transport method that meets the security goals of the participants: in extreme cases, such references have been sent by reading the text characters from the file over the telephone. Membership in DonutLab consists of having these two references: you need a kiosk to find services, and a mint account to use them. Kiosk and account references are unguessable: you must explicitly receive them in order to communicate with the services they authorize.
- The researcher launches his program, giving the program the references to the account and the kiosk.
- The program goes to the kiosk and requests a list of SliverServers that are currently active.
- The program sends his mobile sliver code to each SliverServer along with payment for the service of executing the code and sending messages on the code's behalf.
- The slivers begin execution in the secure distributed DonutLab context.

Each of the basic components described above is embodied in 3-5 pages of source: typically 1 page of setup and configuration code, 2 pages of code for the actual service, and another page of documentation. All of the code is open source and available for download from www.erights.org starting in ***E*** 0.8.27.

Development Highlights

The first milestone in development of the DonutLab was completion of the mint. We had started work at 8AM on Saturday; the mint was complete at 11AM. So it took about 3 hours to build a financial system.

Since most financial systems take somewhat longer than 3 hours to build, this deserves some explanation. First of all, we did not implement any policy in those 3 hours. This lack of policy simplified the problem considerably, but it is not a complete explanation. 2 other reasons for our speed were:

- **A powerful, compact object-capability financial protocol**: The DonutLab mint implements the Waterken IOU protocol [Close04]. This protocol is the culmination of over 20 years of effort [Hardy81, ERTP99, Miller03a] by the object-capability community to build ever more flexible, reliable, and secure electronic rights transfer systems. Since the key to building such systems is to make the system small and simple, the IOU protocol is quite compact despite its expressive power.
- **A true productivity transformation**: An analogy seems appropriate. In 1990, with the C programming language on MS-DOS, it took months to build programs comparable to the Notepad text editor. Today, with memory-safe, garbage-collected, object-oriented languages like Java, a Notepad-like text editor takes only hours. Distributed object capabilities deliver the same kind of productivity transformation for electronic rights transfer that OO delivered for Notepad-like applications. There are several reasons why object-capabilities enable such development speed. The simplest to explain is the built-in encryption/authentication/authorization of the objects distributed across the system. In ***E***, all communications are encrypted all the time, transparently to both programmers and users. Public keys, private keys, single keys, and object authentication are handled inside the language [Miller00]. As a result, the programmer can manipulate remote object references with the same confidence that he has when he receives an object inside a single process: if object A sends to object B a reference to object C in a single process, there is no worry about man-in-the-middle attacks, no concern about DNS spoofing, or fear that the object C received by B is different from the object sent by A. Similarly, the single-process sequential OO programmer does not worry about

synchronization, deadlock, or inconsistency because a variable may be modified in the middle of an operation. With PPOC, these reliability and security guarantees are extended over the network.

Another major milestone was the security review for the SliverServer. To meet our goals, it was crucial that the code executing as a sliver, which was written by someone whom the SliverServer owner should not trust, not have any authorities that could harm the SliverServer owner or any of his assets, including other computers behind his firewall. Yet the sliver cannot be sandboxed in the style of the Java applet: the operator of the sliver must be able to dynamically confer additional authorities of his own to the sliver. In the Doughbot, for example, the Doughbot owner must be able to confer to the sliver authority to communicate with a kiosk, which immediately breaks the confinement demands of the Java applet sandbox.

We held a security review of the SliverServer to ensure that the SliverServer owner was safe while the sliver operator was empowered. That review was shorter than the following explanation of why it was short. Because the review was non-adversarial (the reviewer trusted the developer to tell the truth about related sections of the program, sections that would exhibit dramatic bugs if *accidentally* coded incorrectly), the review came down to a careful analysis of the following line:

```
def bootSliver := sliverMaker <-
    runSliver(bootMakeSturdyRef,
              bootTimeMachine)
```

In an object-capability system, there is no *ambient authority*, i.e., all authority is *denied by default*. Objects are not born with access to the file system, windowing system, or network. As a consequence, in a security review you only have to look for explicit lines of code granting improper authority. The line of code above is the line that constructs the new sliver. The only authorities the sliver receives from the SliverServer are the "bootMakeSturdyRef" and the "bootTimeMachine". Given an understanding of sturdyRefs and timeMachines, one can quickly draw a conclusion about their risks in the context of the threat models and goals of the participants. The speed and ease of object-capability security reviews like this one has been observed in earlier work [Wagner02].

The promise pipelining architecture served principally to ensure no deadlock occurred. The architecture did eliminate many latency-intense round trips (for reasons similar to [Liskov88]), but since DonutLab's goals are generally insensitive to latency this was not a sought-for effect, but rather a side-effect of development in a promise pipelining environment. As noted earlier, datalocks are possible but rare, and none occurred in the development of DonutLab. For a simple example of promise pipelining in action, let us analyze the following line of code. This is the line executed by the Doughbot to pay the SliverServer to create a new sliver that will run a "doughBit":

```
def doughBit := each <- makeSliver(code,
    account <- offer(payPerSliver))
```

The "`<-`" symbol is the *eventual send*, used in ***E*** to communicate across processes and machines. The program does not wait for an answer to be returned from an eventual send; rather, a *promise* is immediately returned for the eventual answer. The promise can be manipulated much like any other object; notably, messages can be sent to the promise using eventual sends, and when the promise is *fulfilled*, those messages will be delivered. Eventual sends to promises create further promises, leading to chains of promises that will be fulfilled as soon as possible, possibly without any intervening round trips. If something goes wrong, a promise is *broken*, and this brokenness propagates to all dependent promises. (Non-signaling NaNs are similarly contagious, and similarly non-disruptive of pipelining optimizations.)

In this example, the Doughbot tells the SliverServer to make a sliver with a body of code and a payment offer. The Doughbot is not actually holding the payment, however. Rather, the Doughbot sends to the SliverServer a promise for the payment, which is constructed by sending a message to the account to make an offer. The SliverServer and the mint (which owns the account) will set up a secure direct connection and route the money straight to the SliverServer (a wise SliverServer owner will wait for the promise of money to resolve into actual money before creating the new sliver, of course, using the ***E*** when-catch clause). Once these messages to the SliverServer and the account are "in the air", the Doughbot's machine could shut down, and the doughBit would still come to life.

The entire project was accelerated by the use of Causeway, a prototype debugger specifically designed for promise pipelining in the ***E*** environment. Causeway exploits the deterministic properties enabled by promise pipelining to give the developer a message-based view of the system. This contrasts markedly to the process-based view offered by conventional debugging systems. In Causeway, by following the messages, the programmer may ignore the division of the object graph into processes, and follow chains of causality wherever they may go.

Limits of Success

The 72-hour DonutLab does not fulfill all the requirements initially set out. Notably it is neither fully decentralized nor fully persistent. We did not have time to build the DoughChanger, a financial exchange required to enable multiple mints to cooperate in a single DonutLab. A DoughChanger has since been developed. It constitutes about four pages of code, much like the other services: a page of setup and configuration code, 2 pages of code for the actual service, and a page of documentation.

Also the SliverServer is not fully persistent. Specifically, it is unable to revive slivers after a shutdown (or crash, after the loss of power) of the server. So a Doughbot cannot pick up and continue its attack after shutdown of its slivers.

Despite these shortfalls, we still feel that overall the effort was a success. It seems likely to take no more than another three day weekend to fulfill all the original requirements, and to add several additional desirable features identified during development.

Future Work

The striking shortfall of DonutLab in comparison with PlanetLab is its inflexible demands on programming tools. PlanetLab, by using virtual OS's, allows the developer of an application to use any tools that can be found on a Linux platform. The current DonutLab requires that applications be written in ***E***. The most obvious approach to overcoming this limitation is to integrate a virtual machine or OS system with the SliverServer. By building a virtual socket manager for the virtual OS that was in close communication with the SliverServer, one could build a system with all the current security properties (assuming the virtuality of the virtual OS could not be breached). It seems possible that this could be accomplished in a long weekend, if a carefully selected team of PlanetLab and DonutLab developers came together for the mission.

Is DonutLab "just a toy"? It is certainly very small. And it is certainly, after 72 hours, incomplete. But smallness is not in itself a proof of unsuitability. As stated by C.A.R. Hoare, "The unavoidable price of reliability is simplicity." For reliability and security, large systems are the real toys: true success can only be achieved in the small.

Whether DonutLab is "just a toy" or not depends on your goals. Even in its current state, it seems better suited as a platform for some current experiments than PlanetLab itself. Because of its usability, it seems possible that DonutLab would encourage a whole new wave of researchers to become involved. Because of its security, it seems possible that DonutLab could encourage a whole new wave of corporate sponsors to become involved.

Whether to enhance PlanetLab with Donut features, or to enhance DonutLab with PlanetLab features, is the quicker and more effective path to the future, seems a lively and appropriate debate to undertake.

Conclusions

Secure distributed system development need not be distinctively and uniquely painful. A set of proper tools, such as promise pipelined object capabilities, can enable both clear conceptualization and straightforward implementation of such systems. PPOC is the natural extension of object-oriented concepts to the distributed world. It extends across the network all the security and reliability characteristics that OO techniques bring so effectively to bear in single-machine single-process development.

DonutLab also demonstrates how an operating environment can encourage "good" behavior in the participants in a system. The Doughbot was designed as a destroyer of systems. In PlanetLab, the Doughbot would succeed. However, inside the Donut, the effort to attack critical services is transformed into an algorithm for giving away money as fast as possible. The Destroyer

becomes Santa Claus. It makes a stark contrast to the state of the current Internet.

References

[Adams03] Robert Adams, **Distributed System Management: PlanetLab Incidents and Management Tools**. PlanetLab Consortium PDN-03-015

[Close04] Tyler Close, **Waterken IOU Design**. www.waterken.com/dev/IOU/Design/

[Dennis66] J.B. Dennis, E.C. Van Horn. "**Programming Semantics for Multiprogrammed Computations**" *Communications of the ACM*, 9(3):143–155, March 1966.

[Hardy81] Norm Hardy, et al..., **Space Banks {Getting New Pages and Nodes}**. *Gnosis Manual*, Agorics 1981, www.agorics.com/Library/KeyKos/Gnosis/62.html

[ERTP99] Mark S. Miller; "**ERTP: The Electronic Rights Transfer Protocol**". www.erights.org/smart-contracts/#ERTP; 1999.

[Liskov88] Barbara Liskov, Liuba Shrira: **Promises: Linguistic Support for Efficient Asynchronous Procedure Calls in Distributed Systems**. PLDI 1988: 260-267

[Miller88] Mark S. Miller, K. Eric Drexler **Markets and Computation: Agoric Open Systems**. *The Ecology of Computation*, Bernardo Huberman (ed.) Elsevier Science Publishers/North-Holland, 1988.

[Miller00] Mark S. Miller, Chip Morningstar, Bill Frantz; "**Capability-based Financial Instruments**". *Proceedings of Financial Cryptography 2000*, Springer Verlag.

[Miller03a] Mark S. Miller, Marc Stiegler; "**The Digital Path: Smart Contracts and the Third World**"; Markets, Information and Communication. Austrian Perspectives on the Internet Economy; Routledge 2003.

[Miller03b] Mark S. Miller, Jonathan S. Shapiro **Paradigm Regained: Abstraction Mechanisms for Access Control**. Proceedings of Eighth Asian Computing Science Conference Tata Institute of Fundamental Research, Mumbai India, edited by Vijay Saraswat. Springer Verlag

[Pai] Vivek S. Pai, Limin Wang, KyoungSoo Park, Ruoming Pang, Larry Peterson, **The Dark Side of the Web: An Open Proxy's View**

[Peterson02] Larry Peterson, Tom Anderson, David Culler, and Timothy Roscoe**, A Blueprint for Introducing Disruptive Technology into the Internet**. *Proceedings of ACM HotNets-I Workshop, Princeton, New Jersey, USA, October 2002*

[Stonebraker94] Michael Stonebraker, Robert Devine†, Marcel Kornacker, Witold Litwin°, Avi Pfeffer, Adam Sah, and Carl Staelin, **An Economic Paradigm for Query Processing and Data Migration in Mariposa**. *Proceedings of 3rd International Conference on Parallel and Distributed Information Systems*, Austin, TX, USA, 28-30 Sept. 1994. Los Alamitos, CA, USA: IEEE Comput. Soc. Press, 1994. p. 58-67.

[Wagner02] David Wagner & Dean Tribble, **A Security Analysis of the Combex DarpaBrowser Architecure**. www.combex.com/papers/darpa-review/index.html.