EDITORIAL

Musings

RIK FARROW

Rik is the editor of *;login:*.
rik@usenix.org

A frontal attack on idleness. That's how I see distributed systems: a way to keep systems as busy as possible. Decades ago, computers were much too expensive to be allowed to sit idle. Today, computers are immensely more powerful, less expensive, and often set up in enormous clusters, but we still need to keep them busy.

Letting a modern system sit idle wastes resources, even if CPUs have gotten more efficient at doing nothing. Hard disks will continue to spin, DRAM will still require dynamic refreshing, and the power supplies and other hardware will continue to turn electricity into heat, whether a system is busy or not. Certainly a busy system will use more energy, but not a lot more.

And all the while a system is sitting there idle, it's becoming obsolete. No wonder there has been such a focus on distributed systems.

History

While researching for this issue, I took a look at the table of contents for Andy Tanenbaum's 1994 edition of *Distributed Systems*. I was quickly reminded that the distributed systems that come to mind today have their roots in the past, going back as far as the diode and tube-based DYSEAC in 1954. The DYSEAC not was actually distributed, but the potential was there and was discussed. The DYSEAC was perhaps the first mobile computer, housed inside a truck.

We need to skip ahead many years before we begin to see functioning distributed systems. While there were parallel systems built earlier, the earliest commercial distributed system was the Apollo AEGIS operating system in 1982, later Apollo Domain. Apollo used Ethernet to share storage and provide security, processing, and even signed software updates. Sun's "the network is the computer" occurred because Sun was attempting to build an Apollo-like system on top of BSD UNIX.

By the end of the 1980s, two well-known research distributed systems had been implemented: Amoeba and Sprite. Both presented single system images, in that a person sitting at any terminal could be using resources on any of the connected systems transparently. Sprite could even migrate processes from one system to another, while both systems used network file systems.

In the early 1990s, largely as a result of the UNIX wars [1] that began when Sun and AT&T decided to unite SunOS (BSD 4.2 to 4.3) and System V UNIX, the Distributed Computing Environment (DCE [2]) was born. The bastard offspring of Apollo Domain (then owned by HP) and the Andrew File System, DCE was designed to provide many of the features of Apollo Domain, but be able to do so across a network consisting of systems from different vendors. Although the Wikipedia page mentions just three major components, getting DCE to run required running six different services, starting them in the correct order, and was not easy to do. And there was no single reference implementation, so with each vendor writing its own version, interoperability was only experimentally achieved.

The Common Object Request Broker Architecture (CORBA [3]) actually has been more successful than DCE. CORBA supports data sharing and remote processing using an interface definition language (IDL) to create abstractions of objects or data structure in many programming languages. The use of an IDL goes back to work Sun did on remote procedure calls (and likely earlier) as a mechanism for sharing data portably. And each participating host needs to run just one service, the request broker, making it simpler to operate than DCE.

Today, we have truly massive distributed systems. These systems use Java programs as their middleware, something that really puzzled me when I first heard of this. Why Java, I thought, when earlier distributed systems, like Amoeba and Sprite, were microkernel-based for performance and efficiency?

MapReduce focuses on the processing of bulk, line-oriented data, where the latency of disk I/O is the big limiting factor, not the processor. The ease of programming in Java greatly outweighed any performance issues through the use of Java. Systems like Cassandra are very successful, and stable, partially because of their reliance on Java.

Idleness

And what does any of this have to do with idleness? In the cases of Amoeba and Sprite, a workstation user could enter a command that would be executed on any available system. Amoeba included a parallel Make (Amake) for compiling in parallel, the better to use resources.

For datacenters, the cost of power and cooling are second only to the costs of servers [4], so keeping those servers busy maximizes efficiency. As servers will only last three to five years before being replaced, letting them sit idle is another form of waste.

There are other ways of using idle servers, the most popular being clouds running VMs. In some ways these too are distributed systems, as they are relying on distributed storage, authorization, and control systems. But cloud systems are not considered distributed systems.

The Lineup

We begin this issue with two articles related to distributed computing. Matthew Grosvenor, Malte Schwarzkopf, Ionel Gog, and Andrew Moore preview their NSDI paper [5] about a technique and software they've developed to reduce network latency for latency-sensitive applications. They first explain the problems caused by queueing, examine other solutions, explain how their simple solution works, then provide some data to back this up. I think their software may be a hit in datacenters.

Ben Pfaff and a large group of people from VMware write about some software that is already a hit. Open vSwitch [6] is the most widely used virtual switch in cloud environments, and runs on all major hypervisor platforms. The authors describe the architecture of vSwitch and techniques that have been used to both improve performance and increase efficiency of this open source software.

I decided to interview Andy Tanenbaum for this issue. Andy has published two editions of his very successful book about distributed systems, but what I was initially interested in was Amoeba, a distributed system designed by Andy and his students in the late '80s. I also wanted to take advantage of the interview to ask some questions about some of his other projects, and to get his opinion on the acceptance of microkernels.

We also have four articles focused on aspects of system administration. Tim Bradshaw leads off with an article about the living dead. These zombies are old systems that you cannot stop supporting even when the official support for the platform has long been deceased. Tim contrasts nimble and fast-growing organizations with ones burdened with decaying body parts, and what this means for system administrators and security.

I convinced Adam Moskowitz to write about his experience creating a testing framework for shell scripts. Adam does a great job making clear why you don't just want, but need, testing for all your scripts. Adam also shares the framework he built for automating script testing as much as possible. If you find yourself disturbed at the thought of having to add testing to your scripts, I suggest you read Adam's article. Perhaps you will see the light, as well as appreciate the toolset Adam recommends.

Andrew Stribblehill and Kavita Guliani take on the topic of managing incidents. Quite simply, you get to manage incidents, instead of managing crises, when you have previously prepared and practiced how to do so. Stribblehill and Guliani contrast unmanaged incidents (crises management) with managed incidents, and provide the elements of a managed incident process.

Andy Seely shares his own generational theory of sysadmins. I fit into the category that his father does (the first generation of explorers), but I do see his point. Like any system of categorization, Andy's system will not be a perfect fit for anyone, but I found his system to be a useful way of understanding the different types of sysadmins you will encounter.

David Blank-Edelman makes browsers dance in his column. David describes how to use the Selenium framework to remotely control a browser from Perl, allowing you to create repeatable tests for software that will be interacting with Web browsers.

Dave Beazley takes issue with how too many Python programmers handle exceptions. Dave criticizes much of the extreme coding he sees, from over-catching exceptions to ignoring them, while providing really practical advice.

Dave Josephsen expands his discussion of Graphios, a tool that he briefly mentioned in his February 2015 column. Dave has been working with the creator of Graphios (Shawn Sterling) to replace Graphios' Graphite-specific backend with a modular framework, so Graphios can extract data from Nagios and share it with other tools.

Dan Geer borrows from biological survey techniques to suggest analysis techniques that can help us decide whether patching is really improving software over time. While most of us wonder how patching cannot be helping (by removing exploitable code), Dan's focus is on whether the patching is making any difference to the overall security of a specimen of software.

Robert Ferrell takes a humorous look at both system administration and distributed systems. While you might imagine these would be difficult topics for humor, Robert manages to do a good job of it.

USENIX is celebrating its fortieth year, and we are adding some new features to *;login:*. Peter Salus will be writing for *;login:* again, reviving his history column with the story of the UNIX wars. We also have the very first issue of *UNIX News,* the predecessor of *;login:*. Finally, we have included another article from our archives. David Brumley, now an associate professor at CMU, wrote about the rootkits he encountered while working at Stanford University.

We have just two book reviews this month, both by Mark Lamourine. We also have a handful of reports from LISA14.

Conference Reports

Speaking of reports, we, the USENIX Board and staff, have decided that we are going to handle reports differently in the future. In the past, we have worked hard in an attempt to cover every session of every conference, symposium, or workshop that USENIX runs, and have done a fairly good job. While we occasionally managed to cover all sessions—for example, at a well-attended, single-track conference like OSDI—conferences like LISA, with its five tracks, have always been difficult to cover.

We also found ourselves competing with the sound and video recordings made of the sessions. USENIX has been recording sessions for many years now, and providing those recordings to anyone interested as part of our commitment to open access to research. Actual recordings of a presentation are much more accurate than reports, as they reproduce what actually happened instead of the summarizer's version of the presentation. While I personally attended many sessions, I lacked the ability to be in many places at once, no matter how many tracks were happening concurrently. I would use my notes to improve some of the reports I received, but will confess that by the end of two (or three or more) days of note taking, my notes were getting a bit sketchy.

In the future, we will continue to solicit and publish reports that we receive when they meet our standards. We will not go to the lengths we have in the past to round up summarizers in an attempt to cover every session, including those (the majority) that are being recorded. As a result, future issues of *;login:* may be a bit thinner. But USENIX will continue to provide audio and video recordings of as many sessions as possible.

The attack on idleness continues to this day, with even smartphones getting into the act. Most any Web program expects to execute code (JavaScript) with the browser, as well as code on the server. Bitcoin mining malware authors have been using distributed systems since 2011, and in 2014, five apps were removed from Google Play because they were attempting to use the comparatively puny processing power of smartphones to mine Bitcoins [7]. Malware that targets desktops with power to spare has been modestly more successful as conscripted miners.

The Internet of Things will continue the movement toward distributed systems, with each Thing doing more than simply collecting and forwarding data, like a FitBit or a Nike shoe. To help with this process, and to leverage the large number of programmers at home in Windows environments, Microsoft has announced that they will be releasing a version of Windows 10 for the Pi 2 [8]. While I welcome the news, I do feel a bit of trepidation, and hope that this is a securely stripped-down version of Windows. In the future, even Things will be kept busy.

References

[1] UNIX wars: http://en.wikipedia.org/wiki/Unix_wars.

[2] DCE, pretty decent reference: http://en.wikipedia.org/wiki/Distributed_Computing_Environment.

[3] CORBA: http://en.wikipedia.org/wiki/Common_Object_Request_Broker_Architecture.

[4] James Hamilton's blog, Perspectives, September 18, 2010: http://perspectives.mvdirona.com/2010/09/18/OverallDataCenterCosts.aspx.

[5] Matthew P. Grosvenor, Malte Schwarzkopf, Ionel Gog, Robert N. M. Watson, Andrew W. Moore, Steven Hand, and Jon Crowcroft, University of Cambridge, "Queues Don't Matter When You Can JUMP Them!"; NSDI '15: https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/grosvenor.

[6] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, Keith Amidon, Martin Casado, VMware, Inc., "The Design and Implementation of Open vSwitch"; NSDI '15: https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/pfaff.

[7] http://www.theregister.co.uk/2014/04/25/yes_there_is_now_bitcoinmining_malware_for_android/.

[8] Windows 10 for Raspberry Pi 2: https://dev.windows.com/en-us/featured/raspberrypi2support.

Letter to the Editor

I read with great interest your interview with Dan Farmer. A strong character, who loves to create things, but wants no part of "productizing"! His tale about Symantec and outsourcing security products made me wonder how that company assesses the honesty of its wares and brought to mind the genre of spam that offers to help rid one's machine of malware.

The piece also brought to mind some bits of ancient history:

COPS definitely wasn't the earliest vulnerability scanner, though it may have been the earliest to be publicly distributed. In fact the README file that comes with COPS says that sysadmins had been in the habit of rolling their own bits and pieces of it before. For one, Fred Grampp—who together with Robert Morris senior wrote a classic paper on UNIX security—had distributed a quite comprehensive security-sniffing suite within Bell Labs; unfortunately, I can't remember its name. The company's first computer-security task force, which I chaired in 1982, was able to back up its warnings with real data, thanks much to Fred.

I well remember the Morris worm. A bunch of people gathered in the research UNIX lab, watching it beat on the gate as we followed its progress by phone contact with other sites across the country. Peter Weinberger, in particular, spent much of the time on the phone with CMU's Software Engineering Institute, whose CERT division today proudly says, "We were there for the first Internet security incident and we're still here 25 years later." That incident led to the formalization of CERT's national role, which in turn provided Dan's first job.

The worm didn't get into Bell Labs, because Dave Presotto recoiled from installing the inscrutable Sendmail system and decided to roll his own. Then, in a reverse twist, he named it after a poison tree. If Dave put into Upas anything like Sendmail's trojan horse (a back door intended for diagnosing troubles reported by confused sysadmins), nobody has found it yet.

Doug McIlroy
doug@cs.dartmouth.edu