;login:

SPRING 2020 VOL. 45, NO. 1

usenix®
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

UPCOMING EVENTS

SREcon20 Americas West
March 24–26, 2020, Santa Clara, CA, USA
www.usenix.org/srecon20americaswest

HotEdge '20: 3rd USENIX Workshop on Hot Topics in Edge Computing
April 30, 2020, Santa Clara, CA, USA
www.usenix.org/hotedge20

OpML '20: 2020 USENIX Conference on Operational Machine Learning
May 1, 2020, Santa Clara, CA, USA
www.usenix.org/opml20

PEPR '20: 2020 USENIX Conference on Privacy Engineering Practice and Respect
May 11–12, 2020, Santa Clara, CA, USA
www.usenix.org/pepr20

SREcon20 Asia/Pacific
June 15–17, 2020, Sydney, Australia
www.usenix.org/srecon20apac

2020 USENIX Annual Technical Conference
July 15–17, 2020, Boston, MA, USA
www.usenix.org/atc20

Co-located with USENIX ATC '20

HotCloud '20: 12th USENIX Workshop on Hot Topics in Cloud Computing
July 13, 2020
www.usenix.org/hotcloud20

HotStorage '20: 12th USENIX Workshop on Hot Topics in Storage and File Systems
July 13, 2020
www.usenix.org/hotstorage20

SOUPS 2020: Sixteenth Symposium on Usable Privacy and Security
August 9–11, 2020, Boston, MA, USA
Co-located with USENIX Security '20
www.usenix.org/soups2020

29th USENIX Security Symposium
August 12–14, 2020, Boston, MA, USA
Co-located with SOUPS 2020
www.usenix.org/sec20

Co-located with USENIX Security '20

WOOT '20: 14th USENIX Workshop on Offensive Technologies
August 10–11, 2020
www.usenix.org/woot20

CSET '20: 13th USENIX Workshop on Cyber Security Experimentation and Test
August 10, 2020
www.usenix.org/cset20

ScAINet '20: 2020 USENIX Security and AI Networking Summit
August 10, 2020
www.usenix.org/scainet20

FOCI '20: 10th USENIX Workshop on Free and Open Communications on the Internet
August 11, 2020
www.usenix.org/foci20

HotSec '20: 2020 USENIX Summit on Hot Topics in Security
August 11, 2020
www.usenix.org/hotsec20

SREcon20 Europe/Middle East/Africa
October 27–29, 2020, Amsterdam, Netherlands
www.usenix.org/srecon20emea

OSDI '20: 14th USENIX Symposium on Operating Systems Design and Implementation
November 4–6, 2020, Banff, Alberta, Canada
Sponsored by USENIX in cooperation with ACM SIGOPS
Abstract registrations due May 5, 2020
www.usenix.org/osdi20

LISA20
December 7–9, 2020, Boston, MA, USA

SREcon20 Americas East
December 7–9, 2020, Boston, MA, USA

USENIX Open Access Policy

USENIX is the first computing association to offer free and open access to all of our conference proceedings and videos. We stand by our mission to foster excellence and innovation while supporting research with a practical bias. Please help us support open access by becoming a USENIX member and asking your colleagues to do the same!

www.usenix.org/membership

www.usenix.org/facebook twitter.com/usenix www.usenix.org/linkedin www.usenix.org/youtube

;login:

SPRING 2020 VOL. 45, NO. 1

usenix® THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

EDITOR
Rik Farrow

MANAGING EDITOR
Michele Nelson

COPY EDITORS
Steve Gilmartin
Amber Ankerholz

PRODUCTION
Arnold Gatilao
Ann Heron
Jasmine Murcia
Olivia Vernetti

TYPESETTER
Star Type

USENIX ASSOCIATION
2560 Ninth Street, Suite 215
Berkeley, California 94710, USA
Phone: +1 510.528.8649
login@usenix.org

www.usenix.org

;login: is the official magazine of the USENIX Association. *;login:* (ISSN 1044-6397) is published quarterly by the USENIX Association, 2560 Ninth Street, Suite 215, Berkeley, CA 94710, USA.

A one-year print subscription to *;login:* magazine is available only to USENIX Association members at the Sustainer level and higher. Periodicals postage is paid at Berkeley, CA, USA, and additional mailing offices.

POSTMASTER: Send address changes to *;login:*, USENIX Association, 2560 Ninth Street, Suite 215, Berkeley, CA 94710, USA.

EDITORIAL

Musings

RIK FARROW

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

I read Brian Kernighan's latest book recently, and many things in it struck chords with me. While the book was mostly about UNIX history, it was what Brian wrote about the influence that UNIX had on the development of computers, programming, and even printing that grabbed my attention.

For example, Brian pointed out that computers had been highly customizable machines that generally ran programs that were terribly inflexible. If you wrote files to disk using one program, you could only use that program, or a closely related one, to manipulate those files. UNIX, by comparison, uses byte streams for files, ones that can be opened by any program, even ones that can't do anything sensible with the bytes, but that flexibility is enormous. Think of the old version of `spell`; that was a pipeline that converted a text file to a list of words, sorted those words, ran `uniq` on them, and then compared the results to a dictionary. None of those tools is unique to a spell-checking program. Today, `spell` is a binary, not a shell script, but you can find a version of the original script in Brian's book.

Computer Architecture

Another idea that caught my attention appears in the book at the bottom of page 128: UNIX and C had a large impact on computing hardware in the 1980s and 1990s. Most successful instruction set architectures were well matched to C and UNIX.

I certainly never really thought about that back when I was working with UNIX workstations in the late 1980s. I know I worked on at least a dozen different workstations in that period, mostly various RISC architectures as that was the hotness of the day.

A key feature of all of those instruction set architectures (ISAs) was that instead of being word oriented, all were byte oriented. That may seem too simple, but consider the types that popular programming languages use: very few are tied to a memory-length word. Some types, like float and double, are closely related to actual hardware in the CPU, but things like arrays, strings, different flavors of integers, structures, are all byte, or multiples of bytes, oriented.

I am not certain that UNIX is responsible for this, but UNIX certainly was a huge influence. I was talking about this with Jon Callas, who worked for DEC in the '90s, and he pointed out that DEC's Alpha CPU worked equally well running Ulrix/64, DEC's VMS, and Windows NT (Windows these days). None of these operating systems and their underlying languages were word oriented, although I do wonder about VMS, which was still written in assembler in the 1990s.

CISC vs. RISC

Today, most servers run variants of Intel's ISA, while the world of the small is mainly RISC. That Intel is byte oriented is no mystery: the Intel 8080 CPU had eight-bit registers and a 16-bit address space. Most registers were paired, so could appear as 16 bits wide, but the only register capable of integer arithmetic was the `A` register, and that was eight bits in width.

In the '80s, we thought that RISC was the way of the future, as RISC allowed CPU designs to be simpler. What happened instead was that Intel kept making up for the weakness of CISC through hardware tricks, including converting their CISC ISA into an internal RISC-like ISA. These tricks require more transistors and more energy, meaning there is still a chance that ARM64 may become more popular in server farms and clouds—but I am not betting on it.

The Lineup

I started my search for authors by looking at the accepted papers at SOSP '19 and found two I particularly liked. Neither won best paper awards, but the author of the first article, Abutalib Aghayev, told me that his paper had been downloaded four times as often as the one that did win the best paper award. I think that's because his topic is more pragmatic.

I also liked the paper that this article is based on because it relates well to my Winter 2019 column about file systems. Aghayev and his colleagues at CMU and Red Hat created BlueStore for Ceph. Ceph is a distributed file system and had been relying on existing file systems for block storage. Aghayev et al. wrote BlueStore to work in raw partitions in just two years, while greatly improving performance and adding features unavailable when Ceph nodes ran over file systems like Btrfs and xfs.

The next article is based on a paper by Anish Athalye and his colleagues at MIT that uses a clever design to solve a security problem that had proved intractable. Hardware wallets, such as used to store and transact Bitcoin, have proven to be vulnerable to attacks, and Athalye fixed this by using two small processors and *reset-based switching,* so that multiple programs can be run on a hardware wallet but be unable to interfere or attack other programs and their data.

Next up we have two articles about AI/ML. Jessica Cussins Newman and Rajvardhan Oak write about ethical considerations for companies and researchers working with AI. The authors present a balanced and thoughtful look at the impacts AI will have on politics, justice, and human rights. L Jean Camp introduced me to the authors, and I am happy to extend our series about ethics with this article.

Nisha Talagala and Joel Young, the co-chairs of OpML '20, tell us about what they learned from the first OpML conference and explain what they expect will come out of the second conference in May 2020. The authors point out that ML differs from earlier computing paradigms, echoing Newman and Oak when it comes to ethical considerations, but also that AI/ML is different operationally.

Switching to SRE/Sysadmin, Luis Mineiro explains how Zalando, Europe's largest online fashion platform, has learned to deal with paging. In an age of distributed systems, when SLIs (service level indictors) show something has gone wrong, you only want to page the people responsible for the sub-system causing the slowdown or outage. And this can be trickier than it might seem.

Todd Palino explains a system for organizing work called "Getting Things Done." GTD is based upon a book, but Palino shares his own experience as well as tools that can be used to support the process. Just about everyone can benefit from learning about and, better, using GTD.

Jaime Woo and Emil Stolarsky examine how to choose the best SLIs. They use the analogy of a famous Florida theme park to explain what works best as indicators of customer satisfaction as opposed to choosing less potent indications of success.

I interviewed Mary Ann Horton. I met Mary Ann while at USENIX ATC '19 in Renton, Washington. She was there for the 50th anniversary of UNIX, celebrated at a gathering at the Living Computer Museum (https://livingcomputers.org) in Seattle. Mary Ann tells us a lot about the history of UNIX from a different perspective than Brian's, as she was part of the creation and spread of Netnews and UUCP mail. Mary Ann also has a story to tell about becoming a transgender programmer, beginning her transition while still at Lucent, the owner of Bell Labs.

Laura Nolan has more to say about SLIs and SLOs. Laura was very impressed with the work of MIT Professor Nancy Leveson, based on the keynote she presented at SREcon19 EMEA. Leveson has studied failures and accidents in complex systems, from waterworks to military air-traffic control, and come up with a better method for understanding complex systems. In the first of a two-part column, Laura examines the reference leg of this system, the input that controls the systems we use today, and ties in management's role to SRE.

Peter Norton discusses Python's memory management. Like other systems that must employ garbage collection, Python's design seeks to be as efficient as possible. But that system is generally opaque to programmers using Python, and Peter explains how to look beneath the covers, and he compares Python's GC to Java.

Mac McEniry expands on his column about handling `go` command lines with `cobra` (*;login:* Summer 2019) with `viper`. `viper` handles command-line defaults in a manner most of us should be familiar with, that is, that options used on the command line have priority, followed by the environment, then by defaults from configuration files.

Musings

Dave Josephsen found himself excited by a newish tool. eBPF has been around for some years now, and Dave has awakened to eBPF's promise of getting better and more specific insight into the workings of the Linux kernel. In this, the first of a two-part column, Dave compares an eBPF script to `iostat` for debugging problems with arrays of disks.

Focusing on cybersecurity, Dan Geer takes another look at job prospects under the growing impact of automation. Using data from the US Bureau of Labor Statistics, Dan helps us get real about where job and salary growth have been over the past decade, something that may be helpful if you are looking for a career or getting ready to jump ship to a new career.

Robert Ferrell ponders artificial ethics, the study of how inert electronics may appear, or not appear, to have any ethics at all.

Mark Lamourine has written three book reviews and managed by chance to parallel some of the topics that appear in this issue. Included in his reviews is one about Brendan Gregg's new book on eBPF. I review Brian Kernighan's *UNIX: A History and a Memoir*.

I've often mused about why CPU design appears conservative, meaning that certain aspects appear again and again in designs from many vendors. Sometimes, it's simply because other designs just don't work as well, such as Transmeta and Itanium, both very long instruction word (VLIW) designs. There are other designs, like Alpha and SPARC, that have hung on longer even though their performance isn't as good as what can be done with Intel-style processors.

I have tried to imagine what the ideal CPU design might look like, but the answer to that still lies in the unknowable future. For now, I am grateful for the CPUs that we have today, ones so powerful, and yet efficient, that we can carry them in our pockets. A very, very long road from the PDP 7, with 32 kilobytes of DRAM, that UNIX was written for.

29TH USENIX SECURITY SYMPOSIUM

BOSTON, MA, USA

Co-located Workshops

WOOT '20

14th USENIX Workshop on Offensive Technologies
August 10–11, 2020
Submissions due May 28, 2020
www.usenix.org/woot20

WOOT '20 aims to present a broad picture of offense and its contributions, bringing together researchers and practitioners in all areas of computer security. Offensive security has changed from a hobby to an industry. No longer an exercise for isolated enthusiasts, offensive security is today a large-scale operation managed by organized, capitalized actors. Meanwhile, the landscape has shifted: software used by millions is built by startups less than a year old, delivered on mobile phones and surveilled by national signals intelligence agencies. In the field's infancy, offensive security research was conducted separately by industry, independent hackers, or in academia. Collaboration between these groups could be difficult. Since 2007, the USENIX Workshop on Offensive Technologies (WOOT) has aimed to bring those communities together.

CSET '20

13th USENIX Workshop on Cyber Security Experimentation and Test
August 10, 2020
Submissions due May 19, 2020
www.usenix.org/cset20

CSET '20 invites submissions on cyber security evaluation, experimentation, measurement, metrics, data, simulations, and testbeds. The science of cyber security poses significant challenges. For example, experiments must recreate relevant, realistic features in order to be meaningful, yet identifying those features and modeling them is very difficult. Repeatability and measurement accuracy are essential in any scientific experiment, yet hard to achieve in practice. Few security-relevant datasets are publicly available for research use and little is understood about what "good datasets" look like. Finally, cyber security experiments and performance evaluations carry significant risks if not properly contained and controlled, yet often require some degree of interaction with the larger world in order to be useful.

ScAINet '20

2020 USENIX Security and AI Networking Summit
August 10, 2020
Talk proposals due March 27, 2020
www.usenix.org/scainet20

ScAINet '20 will be a single track summit of cutting edge and thought-inspiring talks covering a wide range of topics in ML/AI by and for security. The format will be similar to Enigma but with a focus on security and AI. Our goal is to clearly explain emerging challenges, threats, and defenses at the intersection of machine learning and cybersecurity, and to build a rich and vibrant community which brings academia and industry together under the same roof. We view diversity as a key enabler for this goal and actively work to ensure that the ScAINet community encourages and welcomes participation from all employment sectors, racial and ethnic backgrounds, nationalities, and genders.

FOCI '20

10th USENIX Workshop on Free and Open Communications on the Internet
August 11, 2020
www.usenix.org/foci20

FOCI '20 will bring together researchers and practitioners from technology, law, and policy who are working on means to study, detect, or circumvent practices that inhibit free and open communications on the Internet.

HotSec '20

2020 USENIX Summit on Hot Topics in Security
August 11, 2020
www.usenix.org/hotsec20

HotSec '20 aims to bring together researchers across computer security disciplines to discuss the state of the art, with emphasis on future directions and emerging areas. HotSec is not your traditional security workshop! The day will consist of sessions of lightning talks on emerging work and positions in security, followed by discussion among attendees. Lightning talks are 5 MINUTES in duration—time limit strictly enforced with a gong! The format provides a way for lots of individuals to share ideas with others in a quick and more informal way, which will inspire breakout discussion for the remainder of the day.

Registration will open in May 2020.

SYSTEMS

File Systems Unfit as Distributed Storage Back Ends

Lessons from 10 Years of Ceph Evolution

ABUTALIB AGHAYEV, SAGE WEIL, MICHAEL KUCHNIK, MARK NELSON, GREG GANGER, AND GEORGE AMVROSIADIS

Abutalib Aghayev is a PhD student in the Computer Science Department at Carnegie Mellon University. He has broad research interests in computer systems, including storage and file systems, distributed systems, and operating systems. agayev@cs.cmu.edu

Sage Weil is the Lead Architect and co-creator of the Ceph open source distributed storage system. Ceph was created to provide a stable, next generation distributed storage system for Linux. Inktank was co-founded by Sage in 2012 to support enterprise Ceph users, and then acquired by Red Hat in 2014. Today Sage continues to lead the Ceph developer community and to help shape Red Hat's overall storage strategy. sweil@redhat.com

Michael Kuchnik is a PhD student in the Computer Science Department at Carnegie Mellon University and a member of the Parallel Data Lab. His research interests are in the design and analysis of computer systems, specifically those involving storage, high performance computing, or machine learning. Before coming to CMU, he earned his BS in computer engineering from the Georgia Institute of Technology. mkuchnik@cmu.edu

For a decade, the Ceph distributed file system followed the conventional wisdom of building its storage back end on top of local file systems. The experience with different file systems showed that this approach always leaves significant performance on the table while incurring significant accidental complexity [2]. Therefore, the Ceph team embarked on an ambitious project to build BlueStore, a new back end designed to run directly on raw storage devices. Somewhat surprisingly, BlueStore matured in less than two years. It outperformed back ends built atop file systems and got adopted by 70% of users in production.

Figure 1 shows the high-level architecture of Ceph. At the core of Ceph is the Reliable Autonomic Distributed Object Store (RADOS) service. RADOS scales to thousands of Object Storage Devices (OSDs), providing self-healing, self-managing, replicated object storage with strong consistency. Ceph's `librados` library provides a transactional interface for manipulating objects and object collections in RADOS. Out of the box, Ceph provides three services implemented using `librados`: the RADOS Gateway (RGW), an object storage similar to Amazon S3; the RADOS Block Device (RBD), a virtual block device similar to Amazon EBS; and CephFS, a distributed file system with POSIX semantics.

Objects in RADOS are stored in logical partitions called *pools*. Pools can be configured to provide redundancy for the contained objects either through replication or erasure coding. Within a pool, the objects are sharded among aggregation units called *placement groups* (PGs). Depending on the replication factor, PGs are mapped to multiple OSDs using CRUSH, a pseudo-random data distribution algorithm. Clients also use CRUSH to determine the OSD that should contain a given object, obviating the need for a centralized metadata service. PGs and CRUSH form an indirection layer between clients and OSDs that allows the migration of objects between OSDs to adapt to cluster or workload changes.

In every node of a RADOS cluster, there is a separate *Ceph OSD* daemon per local storage device. Each OSD processes I/O requests from `librados` clients and cooperates with peer OSDs to replicate or erasure code updates, migrate data, or recover from failures. Data is persisted to the local device via the internal *ObjectStore* interface, which is the storage back-end interface in Ceph. ObjectStore provides abstractions for objects, object collections, a set of primitives to inspect data, and transactions to update data. A transaction combines an arbitrary number of primitives operating on objects and object collections into an atomic operation.

The FileStore storage back end is an ObjectStore implementation on top of a local file system. In FileStore, an object collection is mapped to a directory and object data is stored in a file. Throughout the years, FileStore was ported to run on top of Btrfs, XFS, ext4, and ZFS, with FileStore on XFS becoming the de facto back end because it scaled better and had faster metadata performance [7].

Mark Nelson joined the Ceph team in January 2012 and has 12 years of experience in distributed systems, HPC, and bioinformatics. Mark works on Ceph performance analysis and is the primary author of the Ceph Benchmarking Toolkit. He runs the weekly Ceph performance meeting and is currently focused on research and development of Ceph's next-generation object store. mnelson@redhat.com

Greg Ganger is the Jatras Professor of Electrical and Computer Engineering at Carnegie Mellon University and Director of the Parallel Data Lab (www.pdl.cmu.edu). He has broad research interests, with current projects exploring system support for large-scale ML (Big Learning), resource management in cloud computing, and software systems for heterogeneous storage clusters, HPC storage, and NVM. His PhD in CS&E is from the University of Michigan. ganger@ece.cmu.edu

George Amvrosiadis is an Assistant Research Professor of Electrical and Computer Engineering at Carnegie Mellon University and a member of the Parallel Data Lab. His current research focuses on distributed and cloud storage, new storage technologies, high performance computing, and storage for machine learning. His team's research has received an R&D100 Award and was featured on *WIRED*, *The Morning Paper*, and *Hacker News*. He co-teaches two graduate courses on Storage Systems and Advanced Cloud Computing attended by 100+ graduate students each. gamvrosi@cmu.edu

BlueStore: A Clean-Slate Approach

The BlueStore storage back end is a new implementation of ObjectStore designed from scratch to run on raw block devices, aiming to solve the challenges [2] faced by FileStore. Some of the main goals of BlueStore were:

1. Fast metadata operations
2. No consistency overhead for object writes
3. Copy-on-write clone operation
4. No journaling double-writes
5. Optimized I/O patterns for HDD and SSD

BlueStore achieved all of these goals within just two years and became the default storage back end in Ceph. Two factors played a key role in why BlueStore matured so quickly compared to general-purpose POSIX file systems that take a decade to mature. First, BlueStore implements a small, special-purpose interface and not a complete POSIX I/O specification. Second, BlueStore is implemented in userspace, which allows it to leverage well-tested and high-performance third-party libraries. Finally, BlueStore's control of the I/O stack enables additional features (see "Features Enabled by BlueStore," below).

The high-level architecture of BlueStore is shown in Figure 2. A space allocator within BlueStore determines the location of new data, which is asynchronously written to raw disk using direct I/O. Internal metadata and user object metadata is stored in RocksDB. The BlueStore space allocator and BlueFS share the disk and periodically communicate to balance free space. The remainder of this section describes metadata and data management in BlueStore.

BlueFS and RocksDB

BlueStore achieves its first goal, **fast metadata operations**, by storing metadata in RocksDB. BlueStore achieves its second goal of **no consistency overhead** with two changes. First, it writes data directly to raw disk, resulting in one cache flush [10] for data write, as opposed to having two cache flushes when writing data to a file on top of a journaling file system. Second, it changes RocksDB to reuse write-ahead log files as a circular buffer, resulting in one cache flush for metadata write—a feature that was upstreamed to the mainline RocksDB.

RocksDB itself runs on BlueFS, a minimal file system designed specifically for RocksDB that runs on a raw storage device. RocksDB abstracts out its requirements from the underlying file system in the *Env* interface. BlueFS is an implementation of this interface in the form of a userspace, extent-based, and journaling file system. It implements basic system calls

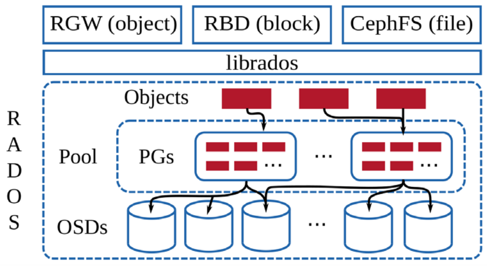


Figure 1: High-level depiction of Ceph's architecture. A single pool with 3× replication is shown. Therefore, each placement group (PG) is replicated on three OSDs.

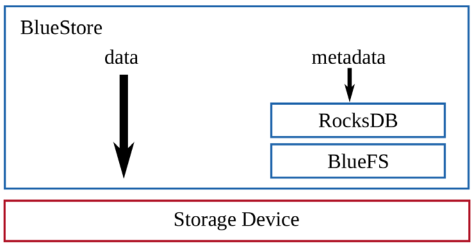


Figure 2: The high-level architecture of BlueStore. Data is written to the raw storage device using direct I/O. Metadata is written to RocksDB running on top of BlueFS. BlueFS is a userspace library file system designed for RocksDB, and it also runs on top of the raw storage device.

required by RocksDB, such as `open`, `mkdir`, and `pwrite`. BlueFS maintains an inode for each file that includes the list of extents allocated to the file. The superblock is stored at a fixed offset and contains an inode for the journal. The journal has the only copy of all file-system metadata, which is loaded into memory at mount time. On every metadata operation, such as directory creation, file creation, and extent allocation, the journal and in-memory metadata are updated. The journal is not stored at a fixed location; its extents are interleaved with other file extents. The journal is compacted and written to a new location when it reaches a preconfigured size, and the new location is recorded in the superblock. These design decisions work because large files and periodic compactions limit the volume of metadata at any point in time.

Metadata Organization. BlueStore keeps multiple namespaces in RocksDB, each storing a different type of metadata. For example, object information is stored in the *O* namespace (that is, RocksDB keys start with *O* and their values represent object metadata), block allocation metadata is stored in the *B* namespace, and collection metadata is stored in the *C* namespace. Each collection maps to a PG and represents a shard of a pool's namespace. The collection name includes the pool identifier and a prefix shared by the collection's object names. For example, a key-value pair `C12.e4-6` identifies a collection in pool 12 with objects that have hash values starting with the six significant bits of `e4`. Hence, the object `012.e532` is a member, whereas the object `012.e832` is not. Such organization of metadata allows a collection of millions of objects to be split into multiple collections merely by changing the number of significant bits. This *collection splitting* operation is necessary to rebalance data across OSDs when, for example, a new OSD is added to the cluster to increase the aggregate capacity or an existing OSD is removed from the cluster due to a malfunction. With FileStore, collection splitting was an expensive operation performed by renaming many directories in a deeply nested hierarchy.

Data Path and Space Allocation

BlueStore is a copy-on-write back end. For incoming writes larger than a *minimum allocation size* (64 KiB for HDDs, 16 KiB for SSDs), the data is written to a newly allocated extent. Once the data is persisted, the corresponding metadata is inserted to RocksDB. This allows BlueStore to provide an **efficient clone operation**. A clone operation simply increments the reference count of dependent extents, and writes are directed to new extents. It also allows BlueStore to **avoid journal double-writes** for object writes and partial overwrites that are larger than the minimum allocation size.

For writes smaller than the minimum allocation size, both data and metadata are first inserted to RocksDB as promises of future I/O and then asynchronously written to disk after the transaction commits. This deferred write mechanism has two purposes. First, it batches small writes to increase efficiency, because new data writes require two I/O operations whereas an insert to RocksDB requires one. Second, it **optimizes I/O based on the device type**: 64 KiB (or smaller) overwrites of a large object on an HDD are performed asynchronously in place to avoid seeks during reads, whereas in-place overwrites only happen for I/O sizes less than 16 KiB on SSDs.

Space Allocation. BlueStore allocates space using two modules: the FreeList manager and the Allocator. The FreeList manager is responsible for a *persistent* representation of the parts of the disk currently in use. Like all metadata in BlueStore, this free list is also stored in RocksDB. The first implementation of the FreeList manager represented in-use regions as key-value pairs with offset and length. The disadvantage of this approach was that the transactions had to be serialized: the old key had to be deleted first before inserting a new key to avoid an inconsistent free list. The second implementation is bitmap-based. Allocation and deallocation operations use RocksDB's merge operator to flip bits corresponding to the affected blocks, eliminating the ordering constraint. The merge operator in RocksDB performs a deferred atomic read-modify-write operation that does not change the semantics and avoids the cost of point queries [8].

The Allocator is responsible for allocating space for the new data. It keeps a copy of the free list in memory and informs the FreeList manager as allocations are made. The first implementation of Allocator was extent-based, dividing the free extents into power-of-two-sized bins. This design was susceptible to fragmentation as disk usage increased. The second implementation uses a hierarchy of indexes layered on top of a single-bit-per-block representation to track whole regions of blocks. Large and small extents can be efficiently found by querying the higher and lower indexes, respectively. This implementation has a fixed memory usage of 35 MiB per terabyte of capacity.

Cache. Since BlueStore is implemented in userspace and accesses the disk using direct I/O, it cannot leverage the OS page cache. As a result, BlueStore implements its own write-through cache in userspace, using the scan-resistant 2Q algorithm. The cache implementation is sharded for parallelism. It uses an identical sharding scheme to Ceph OSDs, which shard requests to collections across multiple cores. This avoids false sharing, so that the same CPU context processing a given client request touches the corresponding 2Q data structures.

Features Enabled by BlueStore

In this section we describe new features implemented in BlueStore. These features were previously lacking because implementing them efficiently requires full control of the I/O stack.

Space-Efficient Checksums

Ceph scrubs metadata every day and data every week. Even with scrubbing, however, if the data is inconsistent across replicas it is hard to be sure which copy is corrupt. Therefore, checksums are indispensable for distributed storage systems that regularly deal with petabytes of data, where bit flips are almost certain to occur.

Most local file systems do not support checksums. When they do, like Btrfs, the checksum is computed over 4 KiB blocks to make block overwrites possible. For 10 TiB of data, storing 32-bit checksums of 4 KiB blocks results in 10 GiB of checksum metadata, which makes it difficult to cache checksums in memory for fast verification.

On the other hand, most of the data stored in distributed file systems is read-only and can be checksummed at a larger granularity. BlueStore computes a checksum for every write and verifies the checksum on every read. While multiple checksum algorithms are supported, `crc32c` is used by default because it is well optimized on both x86 and ARM architectures, and it is sufficient for detecting random bit errors. With full control of the I/O stack, BlueStore can choose the checksum block size based on the I/O hints. For example, if the hints indicate that writes are from the S3-compatible RGW service, then the objects are read-only and the checksum can be computed over 128 KiB blocks, and if the hints indicate that objects are to be compressed, then a checksum can be computed after the compression, significantly reducing the total size of checksum metadata.

Overwrite of Erasure-Coded Data

Ceph has supported erasure-coded (EC) pools through the FileStore back end since 2014. However, until BlueStore, EC pools only supported object appends and deletions—overwrites were slow enough to make the system unusable. As a result, the use of EC pools was limited to RGW; for RBD and CephFS only replicated pools were used.

To avoid the "RAID write hole" problem, where crashing during a multi-step data update can leave the system in an inconsistent state, Ceph performs overwrites in EC pools using two-phase commit. First, all OSDs that store a chunk of the EC object make a copy of the chunk so that they can roll back in case of failure. After all of the OSDs receive the new content and overwrite their chunks, the old copies are discarded. With FileStore on XFS, the first phase is expensive because each OSD performs a physical copy of its chunk. BlueStore, however, makes overwrites practical because its copy-on-write mechanism avoids full physical copies.

Transparent Compression

Transparent compression is crucial for scale-out distributed file systems because 3× replication increases storage costs. BlueStore implements transparent compression where written data is automatically compressed before being stored.

Getting the full benefit of compression requires compressing over large 128 KiB chunks, and compression works well when objects are written in their entirety. For partial overwrites of a compressed object, BlueStore places the new data in a separate location and updates metadata to point to it. When the compressed object gets too fragmented due to multiple overwrites, BlueStore compacts the object by reading and rewriting. In practice, however, BlueStore uses hints and simple heuristics to compress only those objects that are unlikely to experience many overwrites.

Exploring New Interfaces

Despite multiple attempts [5, 9], local file systems are unable to leverage the capacity benefits of SMR drives due to their backward-incompatible interface, and it is unlikely that they will ever do so efficiently [6]. Supporting these denser drives, however, is important for scale-out distributed file systems because it lowers storage costs.

Unconstrained by the block-based designs of local file systems, BlueStore has the freedom of exploring novel interfaces and data layouts. This has recently enabled porting RocksDB and BlueFS to run on host-managed SMR drives, and an effort is underway to store object data on such drives next [1]. In addition, the Ceph community is exploring a new back end that targets a combination of persistent memory and emerging NVMe devices with novel interfaces, such as ZNS SSDs [3].

Evaluation

This section compares the performance of a Ceph cluster using FileStore, a back end built on a local file system, and BlueStore, a back end using the storage device directly. We compare the throughput of object writes to the RADOS distributed object storage.

We ran all experiments on a 16-node Ceph cluster connected with a Cisco Nexus 3264-Q 64-port QSFP+ 40GbE switch. Each node had a 16-core Intel E5-2698Bv3 Xeon 2GHz CPU, 64GiB RAM, 400GB Intel P3600 NVMe SSD, 4TB 7200RPM Seagate ST4000NM0023 HDD, and a Mellanox MCX314A-BCCT 40GbE NIC. All nodes ran Linux kernel 4.15 on Ubuntu 18.04 and the Luminous release (v12.2.11) of Ceph. We used the default Ceph configuration parameters and focused on write performance improvements because most BlueStore optimizations affect writes.

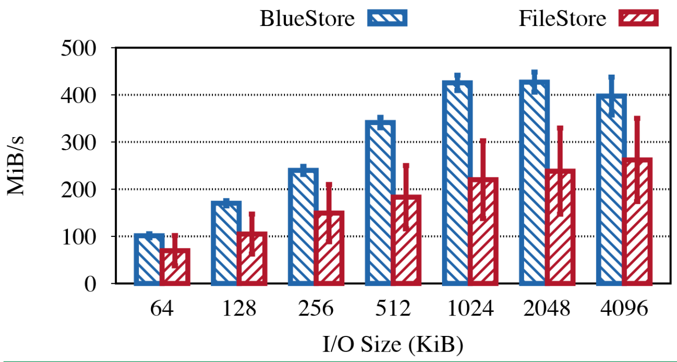


Figure 3: Throughput of steady state object writes to RADOS on a 16-node all-HDD cluster with different sizes using 128 threads. Compared to FileStore, the throughput is 50–100% greater on BlueStore and has a significantly lower variance.

Figure 3 shows the throughput for different object sizes written with a queue depth of 128. At the steady state, the throughput on BlueStore is 50–100% greater than FileStore. The throughput improvement on BlueStore stems from avoiding double writes and consistency overhead.

Figure 4 shows the 95th and above percentile latencies of object writes to RADOS. BlueStore has an order of magnitude lower tail latency than FileStore. In addition, with BlueStore the tail latency increases with the object size, as expected, whereas with FileStore even small-sized object writes may have high tail latency, stemming from the lack of control over writes.

The read performance on BlueStore (not shown) is similar or better than on FileStore for I/O sizes larger than 128 KiB; for smaller I/O sizes, FileStore is better because of the kernel read-ahead. BlueStore does not implement read-ahead on purpose. It is expected that the applications implemented on top of RADOS will perform their own read-ahead.

Conclusion

Distributed file system developers conventionally adopt local file systems as their storage back end. They then try to fit the general-purpose file system abstractions to their needs, incurring significant accidental complexity [4]. At the core of this convention lies the belief that developing a storage back end from scratch is an arduous process, akin to developing a new file system that takes a decade to mature.

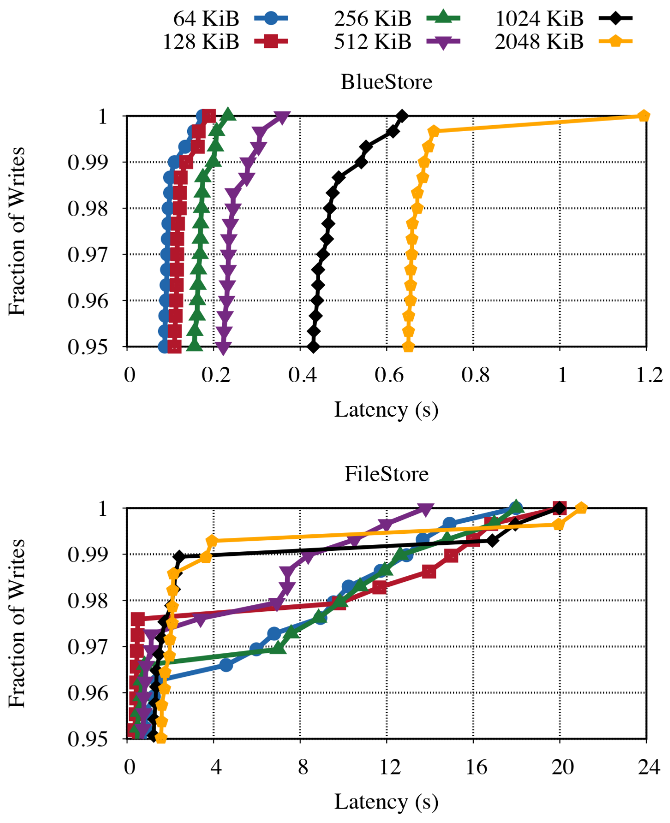


Figure 4: 95th and above percentile latencies of object writes to RADOS on a 16-node all-HDD cluster with different sizes using 128 threads. BlueStore (top graph) has an order of magnitude lower tail latency than FileStore (bottom graph).

Our paper, relying on the Ceph team's experience, showed this belief to be inaccurate. Furthermore, we found that developing a *special-purpose*, userspace storage back end from scratch (1) reclaimed the significant performance left on the table when building a back end on a general-purpose file system; (2) made it possible to adopt novel, backward-incompatible storage hardware; and (3) enabled new features by gaining complete control of the I/O stack. We hope that this experience paper will initiate discussions among storage practitioners and researchers on fresh approaches to designing distributed file systems and their storage back ends.

References

[1] A. Aghayev, S. Weil, G. Ganger, and G. Amvrosiadis, "Reconciling LSM-Trees with Modern Hard Drives Using BlueFS," Technical Report CMU-PDL-19-102, CMU Parallel Data Laboratory, April 2019.

[2] A. Aghayev, S. Weil, M. Kuchnik, M. Nelson, G. R. Ganger, and G. Amvrosiadis, "File Systems Unfit as Distributed Storage Back Ends: Lessons from 10 Years of Ceph Evolution," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles (SOSP '19)*, pp. 353–369.

[3] M. Bjørling, "From Open-Channel SSDs to Zoned Namespaces," 2019 Linux Storage and Filesystems Conference (Vault '19), USENIX Association, 2019: https://www.usenix.org/conference/vault19/presentation/bjorling.

[4] F. P. Brooks Jr., "No Silver Bullet—Essence and Accident in Software Engineering," in *Proceedings of the IFIP 10th World Computing Conference,* 1986, pp. 1069–1076.

[5] D. Chinner, "SMR Layout Optimization for XFS," March 2015: http://xfs.org/images/f/f6/Xfs-smr-structure-0.2.pdf.

[6] J. Edge, "Filesystem Support for SMR Devices," March 2015: https://lwn.net/Articles/637035/.

[7] C. Hellwig, "XFS: The Big Storage File System for Linux," *;login:,* vol. 34, no. 5 (October 2009): https://www.usenix.org/system/files/login/articles/140-hellwig.pdf.

[8] Facebook Inc., RocksDB Merge Operator, 2019: https://github.com/facebook/rocksdb/wiki/Merge-Operator-Implementation.

[9] A. Palmer, "SMRFFS-EXT4—SMR Friendly File System," 2015: https://github.com/Seagate/SMR_FS-EXT4.

[10] Wikipedia, Cache flushing: https://en.wikipedia.org/wiki/Disk_buffer#Cache_flushing.

Save the Dates!

USENIX ATC '20

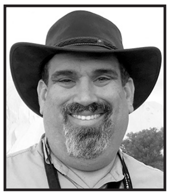
2020 USENIX Annual Technical Conference

JULY 15–17, 2020 • BOSTON, MA, USA

www.usenix.org/atc20

The 2020 USENIX Annual Technical Conference will bring together leading systems researchers for cutting-edge systems research and the opportunity to gain insight into a wealth of must-know topics, including virtualization, system and network management and troubleshooting, cloud and edge computing, security, privacy, and trust, mobile and wireless, and more.

Keynote Speakers



Ethan Miller, *University of California, Santa Cruz*

The Future of the Past: Challenges in Archival Storage

Radhika Nagpal, *Harvard University*

The Pursuit of Collective Intelligence (and Happiness) in Science

Margo Seltzer, *University of British Columbia*

The Fine Line between Bold and Fringe Lunatic

The first USENIX Lifetime Award Keynote!

Registration will open in May 2020.

usenix THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Notary
A Device for Secure Transaction Approval

ANISH ATHALYE, ADAM BELAY, M. FRANS KAASHOEK, ROBERT MORRIS, AND NICKOLAI ZELDOVICH

Anish Athalye is a PhD student in the PDOS group at MIT, working on systems, security, and formal verification. aathalye@mit.edu

Adam Belay is an Assistant Professor of Computer Science at MIT's Electrical Engineering and Computer Science (EECS) department, and a member of the Computer Science and Artificial Intelligence Lab. He received a PhD from Stanford for his work on high performance networking. Recent projects include Shenango, an operating system that improves datacenter efficiency, and Shinjuku, a system that uses fine-grained preemption to reduce tail latency. His current research focuses on the intersection of hardware and software, with an emphasis on improving security and performance. abelay@mit.edu

Frans Kaashoek is the Charles Piper Professor in MIT's EECS department and a member of CSAIL, where he co-leads the Parallel and Distributed Operating Systems Group (http://www.pdos.csail.mit.edu/). Frans is a member of the National Academy of Engineering and the American Academy of Arts and Sciences, and the recipient of the ACM SIGOPS Mark Weiser award and the 2010 ACM Prize in Computing. He was a cofounder of Sightpath, Inc. and Mazu Networks, Inc. His current research focuses on verification of system software. kaashoek@ mit.edu

Notary is a new design for a hardware wallet, a type of security token that is used to protect sensitive transactional operations like cryptocurrency transfers. Notary aims to be more secure than past hardware wallets by eliminating classes of bugs by design and by formally proving the correctness of the key operation used in its implementation. We built a physical prototype of Notary and showed that it achieves functionality similar to existing hardware wallets while avoiding many bugs that affect them.

Hardware wallets, USB keys with a display, buttons, and the ability to run custom code, aim to provide a secure platform for approving transactions such as bank transfers and cryptocurrency transactions. By moving security-critical approval decisions to the device, hardware wallets remove the need to trust relatively complex and bug-prone computers to achieve overall application security. Hardware wallets run multiple applications, which need to be isolated from each other. Existing wallets do this using a traditional operating system design that relies on hardware protection mechanisms like CPU privilege levels and memory protection, but, unfortunately, existing wallets suffer from bugs similar to those that plague traditional computer operating systems.

Notary is a new hardware wallet that aims to avoid many of these bugs by design. Notary achieves strong isolation using *reset-based switching*, along with the use of a physically separate system-on-a-chip for running untrusted code. Notary has a machine-checked proof of the hardware's register-transfer level (RTL) design and software, showing that reset-based switching leaks no state between applications. We built a hardware/software prototype of Notary, along with a number of apps that run on the device, and demonstrated that Notary's design avoids many bugs that affect past hardware wallets.

The Hardware Wallet Paradigm

Users routinely rely on their computers or smartphones to perform and approve security-critical operations. These operations include financial operations, such as bank transfers and cryptocurrency transactions, and non-financial operations, such as system administration tasks like deleting backups or modifying DNS records. The security of these operations relies on the security of the application as well as the underlying platform. Unfortunately, modern computers are inadequate for this purpose because they have complicated software stacks that are full of bugs; even smartphones, often thought to be more secure than PCs, have fallen victim to jailbreaks and malware. On these platforms, buggy or malicious applications might tamper with security-critical operations. Is it possible to achieve security for sensitive transactional operations even when the PC and smartphone are compromised?

Recently, we have seen an increase in the adoption of two-factor authentication (2FA) devices such as Universal 2nd Factor (U2F) tokens, devices that usually come in the shape of a small USB stick and augment the PC to provide additional security for logins. However, these 2FA devices are a bit of a red herring when we are worried about the security of the platform itself, because 2FA devices authenticate the *login* process but not the rest of the interaction with the application. This helps defend against a certain class of attacks, such

Robert Morris is a Professor in MIT's EECS department and a member of the Computer Science and Artificial Intelligence Laboratory. He received a PhD from Harvard University for work on modeling and controlling networks with large numbers of competing connections. His interests include operating systems and distributed systems. rtm@csail.mit.edu

Nickolai Zeldovich is a Professor of EECS at MIT and a member of the Computer Science and Artificial Intelligence Lab. He received his PhD from Stanford University in 2008. Recent projects by Prof. Zeldovich and his students and colleagues include the CryptDB encrypted database, the STACK tool for finding undefined behavior bugs in C programs, the FSCQ formally verified file system, the Algorand cryptocurrency, and the Vuvuzela private messaging system. His current research lies in building practical verified systems. nickolai@csail.mit.edu

as a stolen password: an attacker would not be able to log in to a victim account without the second factor. But it does not help when the platform is compromised: malware on a user's computer waits until the user logs in to a target service (using their U2F token), and then the malware uses the valid session to perform malicious actions.

In contrast, hardware wallets can provide security even when the user's computer is compromised. In the hardware wallet paradigm, an application is refactored to separate out security-critical approval decisions from the rest of the application. An untrusted part of the application runs on the user's PC, while a trusted security-critical *agent* runs on the hardware wallet and is used for approving transactions. The wallet has a display where it shows the user a transaction, and it has buttons to allow the user to confirm or deny the transaction. The approval is required to go through the hardware wallet, and this is generally enforced by requiring a signature with a private key that's stored only in the wallet.

Cryptocurrencies already fit this paradigm where the approval decision is cleanly separated out, and in fact, hardware wallets are already popular with users of cryptocurrencies. For example, users run Bitcoin wallet software on their PC, where they can view their balance, view past incoming and outgoing transactions, and set up transfers, but they cannot actually transfer currency. To send bitcoins, the user crafts a transaction on their PC and sends it to their hardware wallet, which parses the transaction and displays on its screen a human-readable description like "send 1.3 BTC to 1M3K...vUQ7." Only if the user presses a "confirm" button on the hardware wallet does the device sign the transaction, which enables it to be processed by the Bitcoin network.

The paradigm of authenticating transactions on a separate, secure device has gained traction among cryptocurrency users, perhaps due to the high-stakes nature of irreversible transactions. The idea has not yet caught on with more traditional client-server applications like web apps, but there has been some progress in that direction. For example, the Web Authentication API has an extension for transaction authorization, which allows for displaying a prompt string on an authenticator device and receiving confirmation from the user [1].

Hardware Wallets Can Have Bugs Too

With hardware wallets, the PC is removed from the trusted computing base: security depends only on the wallet, which is a big win in terms of security. These devices are much simpler than PCs, and the belief is that while the PC may have been difficult to make secure, the simplicity of wallets allows for more secure designs.

Most hardware wallets today are fixed-function, in the sense that they don't run third-party code: they have built-in support for some fixed set of agents, for example a particular set of cryptocurrencies, and users depend on the firmware vendor to add support for specific applications. This has the obvious downside in terms of usability: when new applications come out, such as a new cryptocurrency, users have to hope that the device manufacturer implements support. The developer of the cryptocurrency has no power to add the support themselves. On the other hand, high-end wallets on the market, such as the Ledger wallet [2], support downloading and running multiple third-party agent applications on the device. This is great for usability, but it adds considerable complexity, requiring that the device be capable of isolating agents from each other, because these third-party agents could be buggy or malicious.

Current devices achieve this by multiplexing the shared hardware between mutually untrusting agents with a traditional operating system using hardware protection mechanisms like CPU privilege modes and memory protection. This leads to the potential for the same kinds of bugs that exist in PC operating systems. And, indeed, existing hardware wallets have suffered from isolation bugs in memory protection configuration, system call

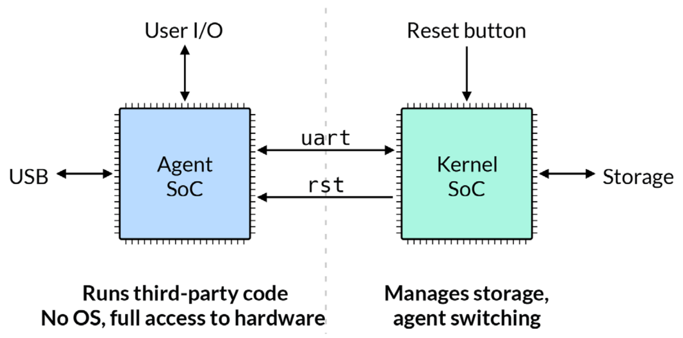


Figure 1: Notary's design physically separates trust domains with an SoC per domain and a simple interconnect between trust domains (reset wire and UART). Untrusted programs are run one-at-a-time on the Agent SoC, which has its state cleared between running agents.

implementations, and driver code [3, 4]. There is also potential for hardware-related bugs: any shared hardware state could potentially be used to infer information about other applications (this is what is happening in attacks like Spectre, for example).

Notary's Approach

Notary is a hardware wallet that aims to avoid by design many of the security issues that affect past wallets. Notary doesn't rely on hardware protection mechanisms like CPU privilege modes or memory protection, and it doesn't have any system calls or even an operating system in the traditional sense. Instead, Notary is built around the idea of achieving isolation by using a dedicated system-on-a-chip (SoC), with its own CPU and memory, to run untrusted programs. Notary runs one program at a time on this chip, and it completely resets this chip (and all of its internal state) when switching between programs, a primitive that's formalized and proven correct in our prototype. Running untrusted code on the dedicated SoC is orchestrated by a separate chip that never runs third-party code.

Figure 1 illustrates Notary's design. The design is structured around physical separation. Notary consists of two security domains, each with its own separate system-on-a-chip (SOC), which includes a CPU, ROM, RAM, and peripherals such as UART. One domain runs the kernel, and one domain runs third-party agent code. The Kernel SoC manages persistent storage and switching between agents; no third-party code ever runs on the Kernel SoC. The Agent SoC, which has no mutable non-volatile storage, runs agent applications one-at-a-time directly on raw hardware (with no OS to protect the hardware). The Agent SoC has direct access to the user I/O path, the buttons and display, as well as access to USB to communicate with the outside world.

In this architecture, after the user chooses an agent to run, it is launched as follows. First, the Kernel SoC resets the Agent SoC and clears all of its internal state. Next, the Kernel SoC reads an agent's code, keys, and data from persistent storage and sends it over the UART; on the other side of the UART, the Agent SoC's bootloader receives the code/data, saves it in RAM, and executes it. At this point, the agent runs directly on top of the hardware on the Agent SoC, not requiring further interaction with the Kernel SoC. The agent has access to everything it needs: its own code and data, the user I/O path, and communication to the outside world. It can do its job, such as displaying a Bitcoin transaction, receiving confirmation from the user, and sending a signed transaction out via USB. Finally, when the agent is done, it has only one way of interacting with the Kernel SoC: a "save and exit" operation, where the agent requests termination, optionally supplying a new persistent state. After this, to run a different agent on the device, the process starts over, beginning with clearing state in the Agent SoC. Notary's separation architecture has analogs for all the operations that hardware wallets generally support: factory-resetting the device, installing/removing agents, and launching agents.

In Notary's design, the decision to connect user I/O and USB directly to the Agent SoC is important for security. An alternative design might connect these to the Kernel SoC, but that would be undesirable because it would introduce the need to have communication between the Agent SoC and Kernel SoC during regular agent operation, adding complexity by requiring a large number of system calls beyond the single save/exit "system call" that Notary supports.

In Notary's design, it is safe to give untrusted code raw access to the user I/O and USB peripherals because the state clearing operation covers peripherals: if a malicious or buggy agent puts the display or USB controller into a bad state, the reset and state clearing operation will fix it. Furthermore, having the display connected to the Agent SoC running potentially untrustworthy code does not introduce the possibility of confusing the user, due to Notary's reset-based workflow. The user switches applications by restarting the entire device, which makes the kernel start a special agent, the application launcher, on the Agent SoC. The user can unambiguously select an agent to run, and after that point, the chosen agent has exclusive control over user I/O until the device is restarted.

With this architecture, Notary achieves isolation between two agents running one after another on the same chip. Running agent code directly on top of raw hardware, using reset as a mechanism to switch agents, obviates the need for a traditional operating system and hardware protection mechanisms, which can be error-prone to program. Performing state clearing, wiping out all state in the Agent SoC between running different agents, ensures that one agent's secrets can't leak to another. Essentially, Notary boils isolation between agents down to state clearing.

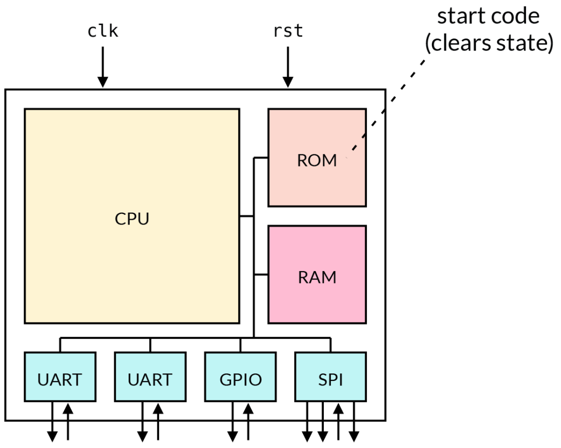


Figure 2: A schematic of Notary's Agent SoC. Carefully written code in boot ROM clears all internal state in the SoC after reset.

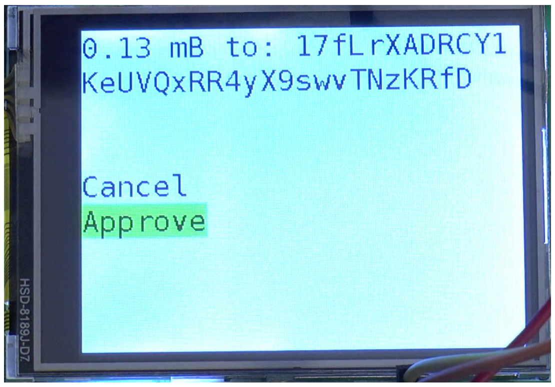


Figure 3: Notary prototype running a Bitcoin wallet agent

State Clearing

Clearing all internal state in a SoC turns out to be challenging, and simple approaches don't work.

At first, we thought that asserting the reset line of an SoC might be adequate. It turns out that ISAs don't guarantee that reset clears internal state; for example, the RISC-V ISA says that the program counter is set to an implementation-defined reset vector, and all other state is undefined [5]. In practice, many chips implement reset such that it only does the minimal work necessary to get the chip going again. For example, on our SoC, asserting the reset line did set the program counter to a well-known value, but it left much state inside the SoC untouched, including in registers, some CPU-internal caches, RAM, and peripherals.

Another approach we considered is power cycling the SoC to clear its internal state. However, research has shown that state inside these chips can persist for minutes without power [6]. Notary applies state clearing before every application switch, so a delay of several minutes to clear state would translate to a delay of several minutes when launching any application, making the device unusable. Furthermore, powering off the SoC for a few minutes provides no guarantees that state is actually cleared.

Provably Correct Software-Based State Clearing

Notary's approach is to use software to clear an SoC's state. The idea is that asserting the reset line resets the program counter, so it can return control to software in boot ROM that can complete the job of clearing all state in the chip, as shown in Figure 2. The idea of having initialization code run on startup is not new, but Notary's boot code is doing something unusual: it's aiming to clear every bit of state *internal* to the SoC, which includes details that don't even exist at the ISA level, such as microarchitectural state. Writing this boot code is a challenge; it's not immediately obvious that writing such code will even be possible. We normally think about code at the abstract machine level, consulting the ISA specification to understand its behavior, but in Notary's case, we need this code to affect internal state.

To help develop this boot code and convince ourselves that it's correct, we built a tool that analyzes an SoC's implementation at the gate level to determine whether the boot code successfully clears all internal state in all situations. The tool takes Verilog code that describes the SoC, converts it to a format compatible with SMT solvers, and then checks whether boot code running on the chip satisfies our correctness property by simulating the circuit symbolically.

Notary's boot code for its simple RISC-V-based SoC, built on the PicoRV32 [7], is formally verified to clear all SoC-internal state correctly using this tool. We are currently working on applying this technique to more complex SoCs.

Prototype

We built a hardware/software prototype of Notary, along with a number of agents that run on the device: a Bitcoin agent and a general-purpose web app approval agent similar to Web Authentication. Figure 3 shows our prototype running the Bitcoin agent in the process of approving a transaction. In our prototype, the heavyweight reset-based approach for launching agents takes about 135 ms, fast enough for interactive use. Of this, 7 ms are spent running the formally verified state clearing code, with most of that time used clearing RAM, and the rest spent copying the agent code/data to the Agent SoC over the relatively slow UART.

Conclusion

Notary is a case study in designing for security. Notary simplifies software (e.g., using reset-based agent switching) and wastes resources (e.g., using physical separation) in order to achieve strong isolation and defense in depth. This separation and reset-based switching eliminates by design classes of bugs that affect traditional user/kernel co-resident designs, including OS bugs, microarchitectural side-channels, and certain hardware bugs. Notary can improve the security of applications where the crucial transaction decision can be succinctly summarized and delegated to a strongly isolated agent running on Notary.

So far, cryptocurrencies have embraced hardware wallets, with significant adoption by users. In the future, we hope to see more applications be refactored to take advantage of the enhanced security that hardware wallets offer.

The full Notary paper is available at https://pdos.csail.mit.edu/papers/notary:sosp19.pdf.

References

[1] W3C, "Web Authentication: An API for Accessing Public Key Credentials," March 2019: https://www.w3.org/TR/webauthn.

[2] "Ledger Hardware Wallets": https://www.ledger.com.

[3] Riscure Team, "Hacking the Ultra-Secure Hardware Cryptowallet," August 2018: https://www.riscure.com/blog/hacking-ultra-secure-hardware-cryptowallet.

[4] C. Guillemet, "Firmware 1.4: Deep Dive into Three Vulner--abilities which Have Been Fixed," March 2018: https://www.ledger.com/2018/03/20/firmware-1-4-deep-dive-security-fixes.

[5] A. Waterman and K. Asanovic, "The RISC-V Instruction Set Manual, Volume II: Privileged Architecture," June 2019: https://riscv.org/specifications/privileged-isa.

[6] A. Rahmati, M. Salajegheh, D. E. Holcomb, J. Sorber, W. P. Burleson, and K. Fu, "TARDIS: Time and Remanence Decay in SRAM to Implement Secure Protocols on Embedded Devices without Clocks," in *Proceedings of the 21st USENIX Security Symposium*, 2012, pp. 221–236: https://www.usenix.org/system/files/conference/usenixsecurity12/sec12-final71.pdf.

[7] C. Wolf, "PicoRV32—A Size-Optimized RISC-V CPU," 2019: https://github.com/cliffordwolf/picorv32.

AI/ML

Artificial Intelligence
Ethics in Practice

JESSICA CUSSINS NEWMAN AND RAJVARDHAN OAK

Jessica Cussins Newman is a Research Fellow at the UC Berkeley Center for Long-Term Cybersecurity where she leads the AI Security Initiative. She is also an AI Policy Specialist for the Future of Life Institute and a Research Advisor with The Future Society. Jessica was a 2016–17 International and Global Affairs Student Fellow at Harvard's Belfer Center, and has held research positions with Harvard's Program on Science, Technology, and Society and the Center for Genetics and Society. Jessica received her master's degree from the Harvard Kennedy School and her bachelor's from the University of California, Berkeley, with highest distinction honors. jessica.cussins@berkeley.edu.

Rajvardhan Oak is a graduate student at the UC Berkeley School of Information. His research interests are security, privacy, and their intersection with machine learning. He obtained his bachelor's degree in computer science from the University of Pune, India. Presently, he is a graduate researcher at the UC Berkeley Center for Long Term Cybersecurity, where he works at the Citizen Clinic and is involved in several public interest projects for low-resource organizations. rvoak@berkeley.edu

This article describes key ethical challenges associated with the design, process, use, and impacts of artificial intelligence. We go beyond naming the problems that have garnered significant attention in recent years, and additionally reference several ongoing efforts to mitigate and manage key ethical concerns. This article is part of a series about ethics intended to encourage ongoing discussion and debate in the research community about ethical considerations that may arise in the course of networking, security, and systems research. We hope that this article will result in researchers as well as industry practitioners being more mindful in their design and use of AI systems.

The role of ethics in AI is sometimes contested, particularly as companies are accused of "ethics washing" in an effort to gain consumer trust while avoiding regulation. Ethics is an important lens for consideration but should not be a substitute for fundamental rights, human rights, or requirements by national and international law. Though this article focuses on AI ethics, it references meaningful intersections with politics, justice, and rights.

Why AI?

From health care to education, from space science to genomic research, AI has revolutionized the way we make decisions. The rise of AI, coupled with the development of computing technology, has allowed us to quickly look at vast amounts of data, discern useful patterns, and use our findings to shape future directions in research or business. Initially intended to be a tool for data analysis and classification tasks, machine learning has now been used to write stories, synthesize images, and even compose music. The desire for AI is understandable; in many cases humans simply cannot match the speed, accuracy, pattern recognition, and large-number-crunching ability of these algorithms. By 2030, AI technologies are expected to contribute $15.7 trillion to the global economy.

What Counts?

The economic promise of AI has led some companies to exaggerate the abilities of their products and services. Companies are able to exploit confusion about what counts as AI because artificial intelligence is an umbrella term, encompassing large sub-fields, including machine learning and deep learning. A simple overarching definition of AI that accounts for the diversity of methodologies actively used is, "a collection of technologies that can enable a machine or system to sense, comprehend, act, and learn" [1]. AI is also considered to be an omni-use technology, meaning that AI technologies have many uses across countless domains, including for good and for ill.

Four Categories

The rise and integration of machine intelligence into the world around us raises numerous ethical challenges, which can be considered to fall within four categories: design, process, use, and impact. The design category includes decisions about what to build, how, and for whom. The process category includes decisions about how to support transparency and accountability through institutional design. The use category includes ways in which AI systems can be used and misused to cause harm to individuals or groups. Lastly, the impact category includes ways in which AI technologies result in broader social, political, psychological, and environmental impacts.

Design

AI systems emerge as a result of numerous human decisions. Many of these may seem innocuous, but they can have profound implications. Most AI systems work by training on large data sets and learning to associate features with outcomes. Machine learning aims to establish a relationship between a target variable and one or more feature variables. It optimizes the parameters of this relation so that the predicted value is as close as possible to the ground truth. However, these systems still make mistakes that a human would never make. Data sets are always imperfect representations of reality and can generate blind spots and biases. Ethical AI is not just judicious use of AI but also thinking carefully about what goes into making these systems.

For example, the tech giant Amazon had been using AI to identify talent and match candidates to jobs since 2014. In 2018, it was discovered that their algorithms systematically discriminated against female candidates. The AI taught itself that the company would prefer male candidates over female ones. For example, according to experts, the system would downgrade candidates from two prominent women's colleges. The algorithm does not know that it is discriminating against women; it simply notices that if it does not select candidates with a certain value (0 or 1) for gender, the results are closer to a defined goal. The problem, in this case, lies in the data that the AI is trained on. Amazon used data that included 10 years of resumes, but only a fraction of them came from women due to women's historical underrepresentation in the technology industry. The AI system, therefore, ranked male candidates over female candidates, since it had seen a greater number of them succeeding.

These failure modes are particularly disturbing when they impact people's livelihoods. In April 2019, over 40,000 residents of Michigan were falsely accused of unemployment fraud based solely on decisions by a machine learning-based computer program. They were forced to repay money, along with substantial penalties. Although the Supreme Court eventually ruled against the governor's office, the fines caused substantial financial burden and even forced some into bankruptcy.

Other design decisions include the composition of engineering teams, and decisions about what technology to build, and for whom. Fewer than 14% of AI researchers are women, and that percentage has decreased over the last 10 years [2]. Racial diversity in AI fares even worse; Google's workforce is only 2.5% Black and 3.6% Latinx, and the percentages at Microsoft and Facebook are similar [3]. The lack of diversity among the teams designing AI systems can also generate blind spots.

For example, AI researcher Joy Buolamwini was a graduate researcher at the MIT Media Lab and found that the facial recognition algorithms she was working with could not "see" her because of her dark skin. She realized that this was not a unique problem; most of the facial recognition community was using the same benchmark data sets for testing the accuracy of models, and the data sets contained extremely limited racial representation. Facial recognition systems were considered to be "accurate" when in fact they were primarily accurate for white men. Joy founded the Algorithmic Justice League to increase awareness about algorithmic bias and develop practices to promote accountability.

Data sets generally reflect historical realities about our world, including structural racism and sexism. When AI systems learn from these data sets, they can then automate and amplify those biases, all while under the veil of technological neutrality. As we rely on algorithmic decision-making in an increasing number of high-stakes environments, including decisions about credit, criminal justice, and jobs, the design and training of the systems should be an area of active consideration.

Process

Just as we need to consider the design of AI systems, we also need to assess the processes in place to support ethical AI. Processes include the implementation of standards and legal requirements, the recognition of principles and best practices, communication with users, and the monitoring of systems' efficacy and impacts. Processes of this kind are necessary to promote transparency and accountability as well as safety.

For example, the utility of massive amounts of data for data analytics and machine learning has contributed to significant privacy breaches. The European Union General Data Protection Regulation (GDPR), which went into force May 2018, is an example of an early regulatory response to help establish data rights and mitigate potential harms from the abuse of personal data. Other data privacy laws have come since, including the California Consumer Privacy Act (CCPA), which went into effect January 2020.

Moreover, it is not common for companies to be forthright about the weaknesses of their models, which can lead to the overestimation of a system's abilities. Unfortunately, we have learned that too much reliance on AI can be dangerous. For example, Uber has been testing their autonomous vehicle technology in Arizona since early 2017. In a shocking incident, a car running in the automatic mode ran a woman over which led to her death. In a similar incident in 2016, a Tesla car running in autopilot mode collided with a truck, leading to the driver's death. Both these cars had human drivers behind the wheel, human drivers who deferred to AI to make the right decision.

Another flaw in AI systems that requires mitigation and monitoring is the susceptibility to *adversarial attacks* [4]. Adversarial examples are those that have been crafted specifically to fool a classifier. Typically, these are constructed by adding a small perturbation to the input. This change is so small that humans cannot identify it; but an algorithm might produce a completely different result. The reason for this is that neural networks, which lie at the heart of most classifiers today, are highly complex and consist of a number of sum functions of logarithms and exponents. As a result, a small change in the input can result in unexpectedly large changes in the output. Research has shown that minor alterations to text, such as dropping a character or capitalizing a letter, can lead to hateful and obscene content being classified as safe. In another example, minute, pixel-level changes to images led a classifier to falsely classify them as facial images.

All machine learning models are capable of making mistakes and being tricked in these ways. And these flaws can be exploited to damaging effect in the real world. For example, researchers have shown how adversarial attacks can be used to confuse medical imaging software, leading to incorrect diagnoses. There are not well-established norms around the mitigation and communication of these risks, and better processes are needed.

Use

Another category of ethical dilemmas associated with AI stems from the technology's broad array of possible uses and misuses. For example, recent advances in AI systems capable of generating synthetic text, audio, and video have beneficial uses, but they can also be used to cause significant harm. Language models can write short stories and poetry, but they can also generate misleading news articles, impersonate others online, automate the production of abusive content, and automate phishing content. *Generative Adversarial Networks* (or GANs) can look at thousands of images of people, learn how faces are constructed, and generate new faces of people who do not exist.

Deepfakes can insert anyone's face into existing video footage, offering a powerful tool for disinformation and information warfare. Doctored videos can quickly spread to millions across social media platforms and can be difficult to detect. Even when quickly proven to be false, doctored videos can have lasting political impact. The rise of deepfakes demands people to be skeptical of what they see, which can breed widespread distrust and corrode democratic processes. For now, however, deepfakes are not widely being used for political destabilization. A study that analyzed thousands of deepfake videos found that the vast majority of deepfakes are being used to create pornographic material, all of which targeted women [5].

Another consequential use of AI is facial recognition technology. In April 2019, it was reported that the Chinese government is using a massive network of facial recognition technology to track and monitor the Uighurs, a largely Muslim minority. The technology has provided unprecedented ability to automate surveillance and repression. Use of the technology has been controversial in the United States as well, where several states have banned the use of facial recognition technology in police body cameras and by law enforcement. Companies have also joined the call for greater regulation of the technology, with Microsoft, for example, calling out the problems of discrimination, privacy abuses, and mass surveillance [6].

AI has also been used to execute military actions. Autonomous weapons are a controversial class of weapons that select and attack targets with limited or no human intervention. Frequently referred to as the third revolution in warfare, after gunpowder and nuclear arms, these weapons may help in reducing human casualties during wars. However, they may also cause terror and destabilization globally; they can be used to conduct assassinations, destabilize nations, and even execute terror attacks on a large scale. In addition to these misuses, these systems are also susceptible to adversarial attacks, biases, and mistakes. Biases in Amazon's systems caused discrimination against women; biases in autonomous weapons can lead to deaths of innocent people.

Impact

AI technologies have economic, political, social, psychological, and environmental impacts that extend well beyond their immediate uses. The long-term impacts to labor markets are one example. The Organization for Economic Cooperation and Development (OECD) estimates that AI and robotics in advanced economies will contribute to radical changes in 32 percent of jobs and fully automate 14 percent of jobs over the next 15–20 years, with disproportionate impacts on low-skilled people and youth [7]. Many countries are now exploring policies to help ease labor transitions for large numbers of people, including retraining programs and social welfare programs.

Another shift that may occur due to AI development is the worsening of economic inequality regionally and between nations. Due to reliance on data and computing infrastructure, AI companies experience network effects, meaning those at the forefront are likely to get increasingly further ahead over time. AI pioneer Kai-Fu Lee has warned that emerging economies are likely to face even greater hurdles as previous pathways to economic growth, for example in China and India, will no longer be available due to the automation of tasks involved in repetitive manual labor of factories and cognitive labor of call centers.

Countries are eager to ensure their economic future and are quickly adopting strategies to generate new talent and innovation. The so-called "race" for AI advancement risks other consequential impacts, however, including international instability and underinvestment in key safety and ethical challenges.

Additionally, AI systems can have long-lasting psychological impacts. For example, e-commerce websites use cookies and demographic data to recommend products to customers. People may feel objectified or unsafe because of the perception that their behavior is being predicted at every step. Most prominent technology platforms also optimize for time spent on their sites, which has led to disturbing advances in "attention hacking" and the facilitation of filter bubbles where people only encounter familiar or provocative content they are likely to engage with. As people communicate more frequently with AI, for example via chatbots, there are also likely to be impacts on human emotions and relationships.

AI also has implications for security infrastructure. Traditionally, security consisted only of the CIA triad; confidentiality, integrity, and authentication. Now, however, there are new loopholes introduced such as susceptibility to adversarial attacks and privacy concerns due to leakage of model parameters. These new vulnerabilities are especially significant for critical infrastructure such as nuclear plants, power grids, and election systems; we now have to ensure security across these additional axes as well.

Lastly, the design and use of AI systems has impacts for the environment. The carbon footprint of training a single AI model has been estimated to result in 284 tons of carbon dioxide—five times the number from an average car over its entire lifetime. Deep learning is particularly energy intensive, as it requires the use of significant computational power for processing vast amounts of data.

Ongoing Efforts

Many institutions are cognizant of the ethical challenges described here and have developed principles to guide their development and use of AI. Notable examples include the Asilomar AI Principles, developed in 2017 through a consultative multi-stakeholder process and signed by thousands of AI researchers and others; Google's AI Principles, developed in 2018, which notably include categories of AI applications that the company will not pursue such as the development of weapons, illegal surveillance, or technologies that would violate international law and human rights; and the Defense Innovation Board's recommendations for AI principles to guide the ethical use of AI by the Department of Defense, published in 2019. Also in 2019, the OECD released AI Principles, which have been endorsed by more than 40 countries as well as by the European Commission and the G20, creating the first intergovernmental standard for the responsible stewardship of AI.

More than two-dozen nations have also released national AI strategies, many of which include discussion of how to manage the ethical implications of AI [8]. For example, France and Singapore have developed policy mechanisms to address ethical issues, including impact assessments and an AI ethics advisory council. In the United States, DARPA has a program dedicated to improving the explainability of AI systems, and the NSF has a program to promote fairness in AI systems [9].

In March 2018, The ACM Future of Computing Academy was sufficiently concerned about the negative impacts of advances in computing that they proposed a change to the peer review process, recommending that peer reviewers require papers to consider both positive and negative impacts. Listing the erosion of privacy and threats to democracy among other concerns, they stated, "we can no longer simply assume that our research will have a net positive impact on the world." The lack of attention to potential negative consequences was described as "a serious and embarrassing intellectual lapse."

Others have proposed different mechanisms for minimizing misuse. For example, when AI company OpenAI developed a new language model capable of generating paragraphs of text based on any prompt, the company described its concerns about how the tool could have negative societal impacts, and announced that they would engage in a staged release plan [10]. OpenAI only released a small version of the model at the outset and then subsequently released larger models over the course of nine months alongside research papers identifying potential social implications and threats. This process was undertaken with the hopes of providing time for more in-depth research into the technology's misuse potential.

Another important mechanism that has been proposed to promote transparency and accountability in AI is the idea of Model Cards. In a 2018 paper titled "Model Cards for Model Reporting," AI researchers proposed that machine learning models should be accompanied by documentation that details their performance characteristics [11]. This is intended to provide benchmarks for evaluation, including whether the model performs consistently across diverse populations, and to clarify intended uses and ill-suited contexts. Model cards are designed to be accessible for both technical and non-technical audiences, and to provide further transparency about how models were trained. Google recently established a web resource to further promote the idea [12].

Algorithmic impact assessments are another tool being used to promote AI accountability [13]. They are intended to examine the use of AI systems; evaluate their impacts on fairness, justice, bias, and other concerns; and to track impacts over time. Additionally, human rights impact assessments, a tool more broadly used for managing the human rights impacts of businesses, projects, and products, are being proposed for use with AI [14]. Predictive policing, targeted surveillance, and disinformation, among other uses of AI, can threaten universal rights. Human rights impact assessments are a key part of the UN Guiding Principles on Business and Human Rights and have been used since 2011. For example, Oxfam America and the Farm Labor Organizing Committee conducted a human rights impact assessment to investigate the state of migrant labor in North Carolina's tobacco industry [15].

Conclusion

AI technologies are not neutral but are created with human goals in mind, taught by human data, and put to use to fulfill human needs; they necessarily have ethical implications. The question is how to increase awareness and establish practices to promote the ethical development of AI that is robust well into the future. Risks of ignoring AI ethics include losing trust from users and the public, as well as pushing away limited talent. The development of ethical AI is a necessary component of sustainable market competition and global leadership.

This article outlined key ethical challenges at stake with artificial intelligence, broken down into four categories of design, process, use, and impact. The article also referenced several ongoing efforts to achieve the goals of ethical AI including principles, strategies, publishing norms, and mechanisms for accountability.

In real-world decision-making scenarios, actors are likely to face tradeoffs between these different considerations. Few ethical guidelines address questions of prioritization, but most organizations will experience the need to decide how to weigh competing values in a given situation. For example, in some cases, there may be a tradeoff between fairness or explainability and accuracy in a machine learning model. Given limited resources, there is also a tradeoff in terms of where to focus.

The need for robust ethical assessment is likely to vary depending on the degree of risk and impact of a given system. However, ethics should not be thought of as an add-on to be considered at the end of production but as a key part of the design process from the outset. Similar to the concept of privacy by design, we need to inculcate the culture of ethics by design. The research community is already at the forefront of many of these debates and is well positioned to play a key role in shaping a positive AI future.

References

[1] "What Is AI Exactly?" *Accenture*, September 21, 2018: https://www.accenture.com/us-en/insights/artificial-intelligence/what-ai-exactly.

[2] "Gender Diversity Crisis in AI: Less Than 14% of AI Researchers Are Women with Numbers Decreasing over the Last 10 Years," Nesta, July 17, 2019: https://www.nesta.org.uk/news/gender-diversity-crisis-ai-less-14-ai-researchers-are-women-numbers-decreasing-over-last-10-years/.

[3] S. M. West, M. Whittaker, K. Crawford, "Discriminating Systems: Gender, Race, and Power in AI," *AI Now*, April 2019: https://ainowinstitute.org/discriminatingsystems.pdf.

[4] L. Huang, A. Joseph, B. Nelson, B. Rubinstein, J. D. Tygar, "Adversarial Machine Learning," Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence, ACM, 2011.

[5] G. Patrini, "Mapping the Deepfake Landscape," *DeepTrace*, July 10, 2019: https://deeptracelabs.com/mapping-the-deepfake-landscape/.

[6] B. Smith, "Facial Recognition: It's Time for Action," Microsoft Blog, December 6, 2018: https://blogs.microsoft.com/on-the-issues/2018/12/06/facial-recognition-its-time-for-action/.

[7] "The Future of Work: OECD Employment Outlook 2019," OECD, 2019: https://www.oecd.org/employment/Employment-Outlook-2019-Highlight-EN.pdf.

[8] J. C. Newman, "Toward AI Security: Global Aspirations for a More Resilient Future," Center for Long-Term Cybersecurity, February 2019: https://cltc.berkeley.edu/wp-content/uploads/2019/02/Toward_AI_Security.pdf.

[9] "Artificial Intelligence for the American People," United States White House, 2019: https://www.whitehouse.gov/ai.

[10] "Better Language Models and Their Implications," *OpenAI*, February 14, 2019: https://openai.com/blog/better-language-models/.

[11] M. Mitchell, S. Wu, A. Zaldivar, P. Barnes, L. Vasserman, B. Hutchinson, E. Spitzer, I. D. Raji, T. Gebru, "Model Cards for Model Reporting," *arXiv*, January 14, 2019: https://arxiv.org/abs/1810.03993.

[12] Google, "Model Cards," 2019: https://modelcards.withgoogle.com/about.

[13] D. Reisman, J. Schultz, M. Whittaker, K. Crawford, "Algorithmic Impact Assessments: A Practical Framework for Public Agency Accountability," *AI Now*, April 2018: https://ainowinstitute.org/aiareport2018.pdf.

[14] "Closing the Human Rights Gap in AI Governance," *ElementAI*, November 2019: http://mediaethics.ca/wp-content/uploads/2019/11/closing-the-human-rights-gap-in-ai-governance_whitepaper.pdf.

[15] Oxfam America and Farm Labor Organizing Committee, "A State of Fear: Human Rights Abuses in North Carolina's Tobacco Industry," 2011: http://hria.equalit.ie/pdf/en/22/A%20State%20of%20Fear.pdf.

The Emerging Practice of Operational ML
USENIX OpML Conference

NISHA TALAGALA AND JOEL YOUNG

Nisha Talagala is the CEO/Founder of Pyxeda. Previously, Nisha co-founded ParallelM and pioneered MLOps, acquired by DataRobot. Nisha also drove the USENIX Conference on Operational Machine Learning, the first conference on production ML. Nisha has 20 years of experience in software, distributed systems, technical strategy, and product leadership. Her prior roles include Fellow at SanDisk and Fellow/Lead Architect at Fusion-io, at Intel as Server Flash/Persistent Memory Lead and at Gear6 as CTO. Nisha earned her PhD at UC Berkeley, holds 69 patents, is a frequent speaker at industry/academic events, and is a contributing writer to several online publications. nishatalagala@gmail.com

Joel leads LinkedIn's Assessments team and co-led LinkedIn's Productive Machine Learning effort. Prior to LinkedIn, he was an Assistant Professor at the Naval Postgraduate School and also at the Air Force Institute of Technology. In addition to academia and over 20 years in the Air Force, he was an Eagle Scout, a hospital janitor, a carnival operator, and a ditch digger. Joel earned his PhD in computer science from Brown University in 2005. His research areas include digital forensics, data mining, and machine learning. He has also published in web search, computational biology, and temporal modeling. joel.d.young@gmail.com

At OpML '19, the first USENIX Conference on Operational Machine Learning, we learned many useful lessons. Moving forward, we expect the same will hold true for the second conference, coming this May 2020. In this article, we discuss some of the pragmatic practices that came out of the first conference.

Machine learning (ML) and its variants such as deep learning (DL) and reinforcement learning are starting to impact every commercial industry. In recognition of the growing need to drive ML into production, and the unique technical challenges therein, USENIX launched OpML in 2019 (Conference on Operational Machine Learning). The first conference dedicated to the operational aspects of machine learning and its variants, OpML is focused on the full life cycle of deploying and managing ML into production [1]. OpML '19 was an energetic gathering of experts, practitioners, and researchers who came together for one day in Santa Clara, CA, to talk about the problems, practices, new tools, and cutting-edge research on production machine learning in industries ranging from finance, insurance, health care, security, web scale, manufacturing, and others [2].

While there were many great presentations, papers, panels, and posters (too many to talk about individually—check out all the details here [2]), there were several emergent trends and themes (previously described here [11]). We expect each of these will expand and become even more prominent over the next several years as more organizations push ML into production and adopt machine learning ops practices to scale ML in production.

Agile Methodologies Meet Machine Learning

Many practitioners emphasized the importance of iteration and continuous improvement to achieving production ML success. Much like software, machine learning improves through iteration and regular production releases. Those who have ML running at scale make it a point to recommend that projects should start with either no Machine Learning or simple Machine Learning to establish a baseline. As one practitioner put it, you don't want to spend a year investing in a complex deep learning solution, only to find out after deployment that a simpler non-ML method can outperform it [3].

Bringing agility to ML also requires that the infrastructure be optimized to support agile rollouts (and rollbacks!). This means that successful production ML infrastructure includes automated deployment, modularity, use of microservices, and also avoiding fine-grained optimization early on [3].

ML-Specific Production Diagnostics because ML Bugs Differ from Software Bugs

Various presentations provided memorable examples of how ML errors not only bypass conventional production checks but can actually look like better production performance. For example—an ML model that fails and generates a default output can actually cause a performance boost!

Detecting ML bugs in production requires specialized techniques like Model Performance Predictors [4], comparisons with non-ML baselines, visual debugging tools [5], and metric-driven design of the operational ML infrastructure. Facebook, Uber, and other organizations experienced with large-scale production machine learning ops, emphasized the importance of ML-specific production metrics that range from health checks to ML-specific (such as GPU) resource utilization metrics [6].

Rich Open Source Ecosystem for All Aspects of Machine Learning Ops

The rich open source ecosystem for model development (with TensorFlow, Scikit-learn, Spark, PyTorch, R, etc.) is well known. OpML showcased how the open source ecosystem for machine learning ops is growing rapidly, with powerful publicly available tooling used by large and small companies alike. Examples include Apache Atlas for governance and compliance, Kubeflow for machine learning ops on Kubernetes, MLflow for life-cycle management, and Tensorflow tracing for monitoring. Classic enterprise vendors are starting to integrate these open source packages to fill solutions (see, e.g., Cisco's support of Kubeflow). Furthermore, web-scale companies are open sourcing the core infrastructure that drives their production ML, such as the ML orchestration tool TonY from LinkedIn [7].

As these tools become more prominent, full end-to-end use cases are also being documented by practitioners, creating design patterns that can be used as best practices by others.

Cloud-Based Services and SaaS Make Production ML Easier

For a team trying to deploy ML in production for the first few times, the process can be daunting, even with open source tools available for each stage of the process. The cloud offers an alternative because the resource management aspects (such as machine provisioning, auto-scaling, elasticity, etc.) are handled by the cloud back end. When accelerators (GPUs, TPUs, etc.) are used, production resource management is challenging. Using cloud services is a way to get started by leveraging the investments made by cloud providers to optimize accelerator usage. Find out more in Ananthanarayanan et al.'s slides at [8].

Cloud deployment can also create a ramp-up path for an IT organization to try ML deployment without a large in-house infrastructure roll out. As discussed by Wenzel and Maurice [9], even on-premise enterprise deployments are moving to self-service production ML models similar to cloud services, enabling the IT organization to serve the production ML needs of multiple teams and business units.

Leverage Expertise from At-Scale Web-Based ML Operations for Enterprise

At-scale experts like LinkedIn, Facebook, Google, Airbnb, Uber, and others, who were the first ML adopters, had to build from scratch all of the infrastructure and practices needed to extract monetary value out of ML. Now these experts are sharing not only their code but also their experiences and hard-won knowledge, which can be adopted for the benefits of enterprise. As the Experts Panel at OpML pointed out [3], the best practices that these organizations follow for ML infrastructure (from team composition and reliability engineering to resource management) contain powerful insights that enterprises can benefit from as they seek to expand their production ML footprint. Experiences from scale ML deployments at Microsoft and others [2] can show enterprises how to deliver performant machine learning into their business applications.

Other end-to-end experiences from at-scale companies [2] showed how business metrics can be translated into ML solutions and the consequent ML solution iteratively improved for business benefit. Finally, organizations facing the unique challenges that edge deployment places on machine learning ops can benefit from learning of scale deployments already in place.

Moving Forward: OpML '20

The goal of the OpML conference is to help develop robust practices for scaling the management of models (i.e., artifacts of learning from big data) throughout their life cycle. Through such practices, we can help organizations transition from manual hand-holding to automated management of ML models in production—the ML version of the move in server operations from "pets to cattle" [12]. Production ML is still a nascent field, and OpML '19 showcased some emerging best practices as described above. New challenges emerge every day, however, such as regulatory concerns brought on by GDPR and CCPA, migrating from legacy infrastructure to cloud, and security attacks on ML systems, just to name a few. OpML '20, to be held in May in Santa Clara, CA, USA, will continue the example set by OpML '19 and be a venue for experts, practitioners, and researchers to discuss, debate, and share the state of the art in Operational ML.

Summary

A great op-ed piece by Michael Jordan in Medium—"Artificial Intelligence: The Revolution Hasn't Happened Yet"—highlighted the importance of an engineering practice for AI [9]. OpML '19, the first Machine Learning Ops conference, illustrated how the ML and AI industry is maturing in this direction, with more and more organizations either struggling with the operational and life-cycle management aspects of machine learning in production, or pushing to scale ML operations and develop operational best practices. This is great news for the AI industry since it is a step further towards generating real ROI from AI investments. OpML '20, following last year's success, will continue to support and bring together the Operational ML community and help realize the long-awaited potential of AI business value. Please join us at OpML '20! [13].

References

[1] OpML '19: https://www.usenix.org/conference/opml19.

[2] OpML '19 Program: https://www.usenix.org/conference/opml19/program.

[3] J. Young, "How the Experts Do It: Production ML at Scale," June 7, 2019: https://www.linkedin.com/pulse/how-experts-do-production-ml-scale-joel-young/.

[4] S. Ghanta, S. Subramanian, L. Khermosh, H. Shah, Y. Goldberg, S. Sundararaman, D. Roselli, N. Talagala, "MPP: Model Performance Predictor" (slides): https://www.usenix.org/sites/default/files/conference/protected-files/opml19_slides_ghantapdf.pdf.

[5] L. Li, Y. Bai, Y. Wang, "Manifold: Model-Agnostic Visual Debugging Tool for ML" : https://www.usenix.org/sites/default/files/conference/protected-files/opml19_slides_li-lezhi.pdf.

[6] Y. Yan, Z. Zweiger, "The Power of Metrics: How to Monitor and Improve ML Efficiency": https://www.usenix.org/sites/default/files/conference/protected-files/opml19_slides_yan.pdf.

[7] J. Hung, "TonY: An Orchestrator for Distributed Machine Learning Jobs" (slides): https://www.usenix.org/sites/default/files/conference/protected-files/opml19_slides_hsu.pdf.

[8] R. Ananthanarayanan, P. Brandt, M. Joshi, M. Sathiamoorthy, "Opportunities and Challenges of Machine Learning Accelerators in Production" (slides): https://www.usenix.org/sites/default/files/conference/protected-files/opml19_slides_ananthanarayanan.pdf.

[9] T. Wenzel and V. Maurice, "Machine Learning Models as a Service" (slides): https://www.usenix.org/sites/default/files/conference/protected-files/opml19_slides_wenzel.pdf.

[10] https://medium.com/@mijordan3/artificial-intelligence-the-revolution-hasnt-happened-yet-5e1d5812e1e7.

[11] https://www.forbes.com/sites/cognitiveworld/2019/07/01/five-trends-in-machine-learning-ops-takeaways-from-the-first-operational-ml-conference/364421147132.

[12] Impact of Data Regulations and Bias on Operational ML panel: https://www.usenix.org/conference/opml19/presentation/panel-uttamchandani.

[13] OpML '20: https://www.usenix.org/conference/opml20.

Are We All on the Same Page?
Let's Fix That

LUIS MINEIRO

Luis's broad background in software engineering includes experience in DevOps, system administration, networking, and more. Luis has been with Zalando since 2013, working with approximately two hundred engineering teams increasing the observability and reliability of the Zalando e-commerce platform, currently heading Site Reliability Engineering.
luis@zalando.de

Industry has defined as good practice to have as few alerts as possible, by alerting on symptoms that are associated with end-user pain rather than trying to catch every possible way that pain could be caused. Organizations with complex distributed systems that span dozens of teams can have a hard time following such practice without burning out the teams owning the client-facing services. A typical solution is to have alerts on all the layers of their distributed systems. This approach almost always leads to an excessive number of alerts and results in alert fatigue. I propose a solution to this problem by paging only the team closest to the problem.

The Age of the Monolith

Many organizations became successful running a monolith. In the age of the monolith we had single, large boxes that did everything—they handled every request. There were some minor evolutions of this basic model, namely for redundancy and availability, but that's not so relevant. What's important—monoliths were simple. They were easy to reason about and easy to monitor.

This was the time of the Ops and Dev silos. The Ops people monitored the hardware and checked whether the monolith process was up. The Devs monitored the requests and the responses.

This approach had its own share of problems, particularly as businesses grew and the approach didn't allow the business to scale further. Microservices have become the solution for those problems.

Modern Microservices

The diagram in Figure 1 is a possible representation of a typical business operation in e-commerce websites—placing an order.

Founded in 2008 in Berlin, Zalando is Europe's leading online fashion platform and connects customers, brands, and partners. It has more than 200 software delivery teams. Organizations such as Zalando can have north of 60 microservices involved in such a business operation, including some so-called legacy ones. Other organizations can actually be simpler or more complex, so mileage may vary. The relevant question is, how do we monitor and alert on this?

The industry came up with new job roles, some call them DevOps, some call them SRE, but the name is not important. We could call them Cupcake Fairies; it doesn't matter. What matters is how we monitor didn't change much, and the new roles didn't change anything. We still check whether boxes are alive, processes are responsive, and individual microservices succeed. Most times, we also check whether responses are fast enough.

When it comes to monitoring, I'd say that we're just monitoring distributed monoliths.

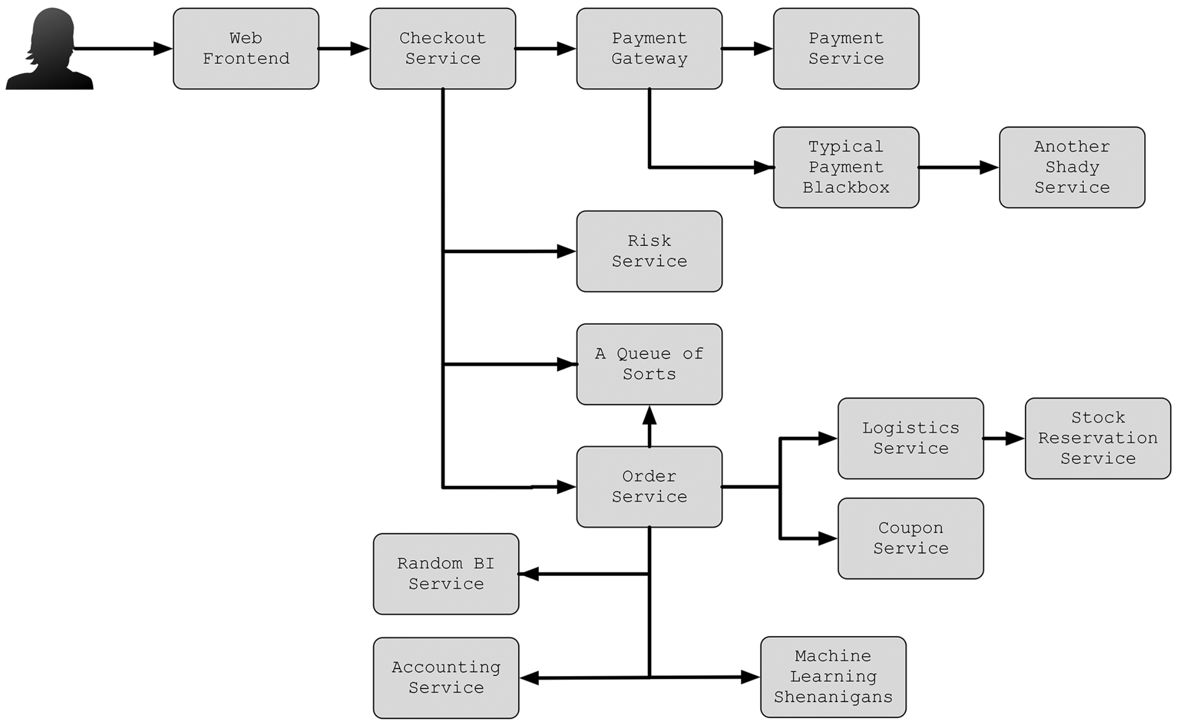


Figure 1: An example set of the microservices involved in fulfilling a customer request. Arrows show the flow through the services and also indicate dependencies.

Problem Statement

What about alerting? What happens when the Accounting Service from the example diagram in Figure 1 has an outage? What almost always happens is that dozens (or hundreds) of alerts come up, making it look like all services failed.

I call this the Christmas Tree effect. Lots of blinking lights, almost the same as Christmas except the happiness level is different, and definitely no one is getting any presents!

This approach almost always leads to an excessive number of alerts and results in alert fatigue. Only one of those teams can actually do something about it—the one operating the Accounting Service.

The alternative to this is to alert on symptoms instead. That's something the industry already accepted—in theory. How would it look if we were alerting on symptoms?

We can measure signals like latency and errors where the Web front end calls the Checkout Service. This is a good place to measure such service level indicators, where the signal-to-noise ratio is optimal and as close as possible to the customer pain.

What happens when alerting on the symptom if the Accounting Service has an outage?

The alert created based on the symptom will be triggered. This looks better. Is there anything wrong with the approach? What happens with this approach if the Payment Service has an outage? The same alert will be up. The team owning the client-facing service, and typically the owner of the alert rule, gets the paging alert *for each and every possible failure in the distributed system!*

This sort of pivoting is a serious problem that hasn't been addressed properly as far as I know. Alerting on all the layers of the distributed system is not healthy, and the alternative, alerting on symptoms, can result in bombing the team owning the client-facing service.

In a Twitter thread [1] early this year, Jacob Scott (@jhscott) brought up the question—"In a 'microservices organization' where teams own specific components/services of a distributed production system, who is responsible for triage/debugging/routing of issues that don't present with a clear owner? And how do they not hate their lives?" Charity Majors' (@mipsytipsy) reply, that I totally agree with, was "alright, this is a damn good question. and tbh i am surprised it doesn't come up more often, because it gets right to the beating heart of what makes any microservices architecture good or bad." This captures the essence of the problem. The so-called "microservices organizations" struggle to figure this out.

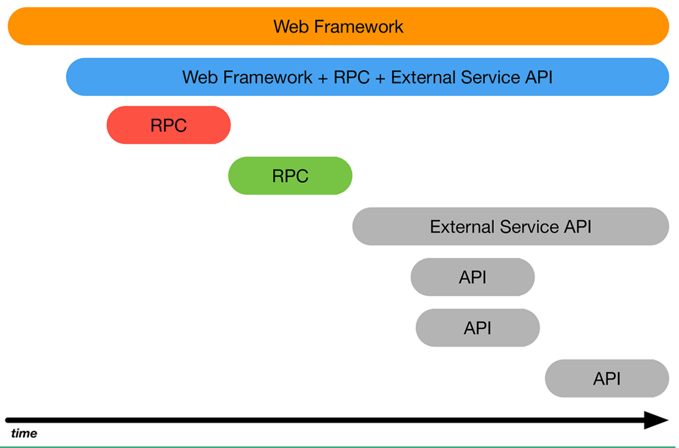


Figure 2: An example trace containing many spans from different microservices

Adaptive Paging

At Zalando we started addressing this problem with a custom alert handler that leverages the causality from tracing and OpenTracing's semantic conventions to page the team closest to the problem. We called it Adaptive Paging.

Five-Minute Introduction to OpenTracing

OpenTracing is a set of vendor-neutral APIs and a code instrumentation standard for distributed tracing. A trace tells the story of a transaction or workflow as it propagates through a distributed system. It's basically a directed acyclic graph (DAG), with a clear start and a clear end—no loops. A trace is made up of spans representing contiguous segments of work in that trace.

You can find a lot more details by checking distributed tracing's origins, namely the Dapper paper [2].

It's worth mentioning that OpenTracing has merged with another instrumentation standard—OpenCensus—resulting in OpenTelemetry. OpenTelemetry will offer backwards compatibility with existing OpenTracing integrations. The concepts and strategy for Adaptive Paging are still valid.

Spans

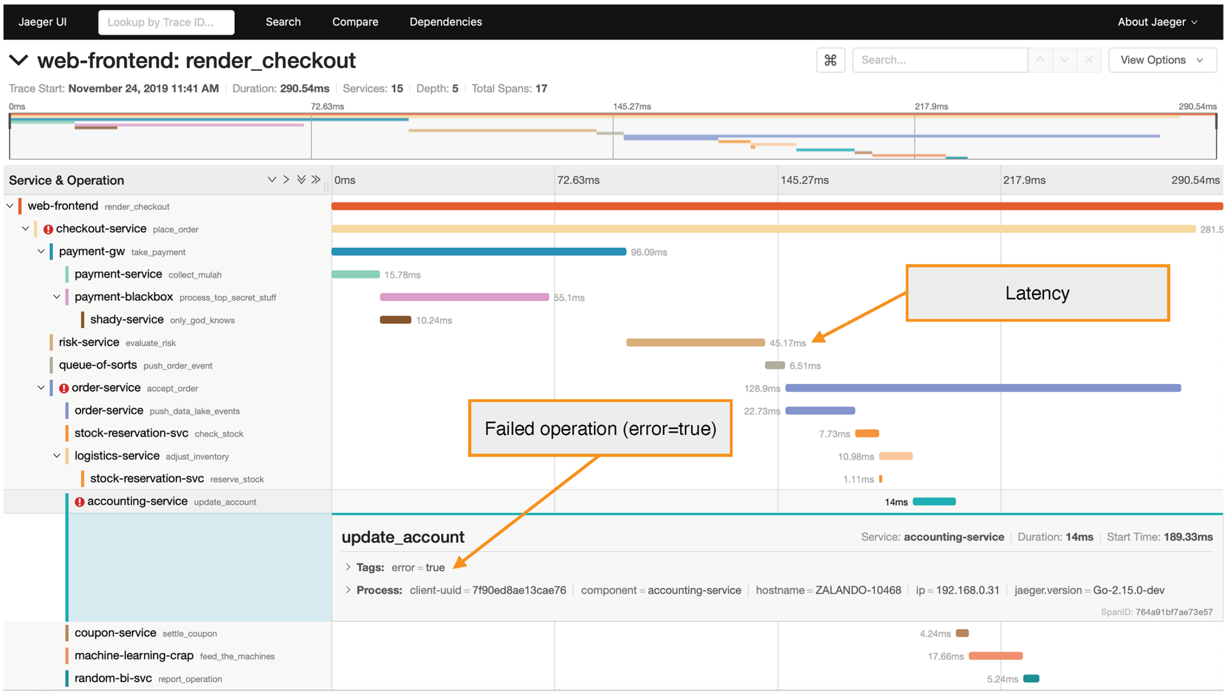
A Span is a named operation which records the duration, usually a remote procedure call, with optional Tags and Logs. This is probably the most important element of OpenTracing. A trace is a collection of spans.

Operations can trigger other operations and depend on their outcome. For example, *place_order* triggers and depends on all the other operations, including *update_account* in the *accounting-service*. This causality is important.

Tags

The other most relevant element from OpenTracing is Tags. A tag is a "mostly" arbitrary key-value pair, where the value can be a string, number, or bool. Every operation can have its own set of tags.

We can consider Tags as metadata that enrich the operation abstraction (the span) with additional context.



```
web-frontend: render_checkout
Trace Start: November 24, 2019 11:41 AM | Duration: 290.54ms | Services: 15 | Depth: 5 | Total Spans: 17

Service & Operation                                 0ms   72.63ms   145.27ms   217.9ms   290.54ms
web-frontend render_checkout
  checkout-service place_order                      281.5
    payment-gw take_payment                         96.09ms
      payment-service collect_mulah                 15.78ms
      payment-blackbox process_top_secret_stuff     55.1ms
        shady-service only_god_knows                10.24ms
    risk-service evaluate_risk                      45.17ms
    queue-of-sorts push_order_event                 6.51ms
    order-service accept_order                      128.9ms
      order-service push_data_lake_events           22.73ms
      stock-reservation-svc check_stock             7.73ms
      logistics-service adjust_inventory            10.98ms
        stock-reservation-svc reserve_stock         1.11ms
      accounting-service update_account             14ms

update_account          Service: accounting-service | Duration: 14ms | Start Time: 189.33ms
> Tags: error = true
> Process: client-uuid = 7f90ed8ae13cae76 | component = accounting-service | hostname = ZALANDO-10468 | ip = 192.168.0.31 | jaeger.version = Go-2.15.0-dev
                                                                                    SpanID: 764a91bf7ae73e57

      coupon-service settle_coupon                  4.24ms
      machine-learning-crap feed_the_machines       17.66ms
      random-bi-svc report_operation                5.24ms
```

Figure 3: Screen capture of a trace in the open source tracing tool Jaeger [3]

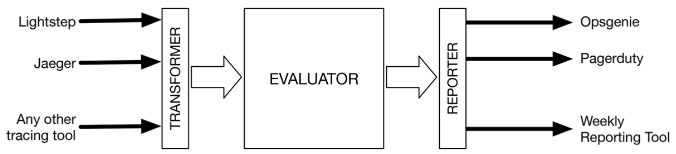


Figure 4: Adaptive Paging components and data flow

Semantic Conventions
OpenTracing's semantic conventions establish certain tag names and their meanings. The existing conventions are strong enough to set certain expectations and enable tools to apply different behaviors when analyzing the tracing data.

OpenTracing Monitoring Signals
OpenTracing can provide, implicitly, measurements for latency and throughput (number of operations over a certain time period). Through the semantic conventions it's also possible to measure errors, by checking the spans with the *error* tag set to the Boolean value *true*.

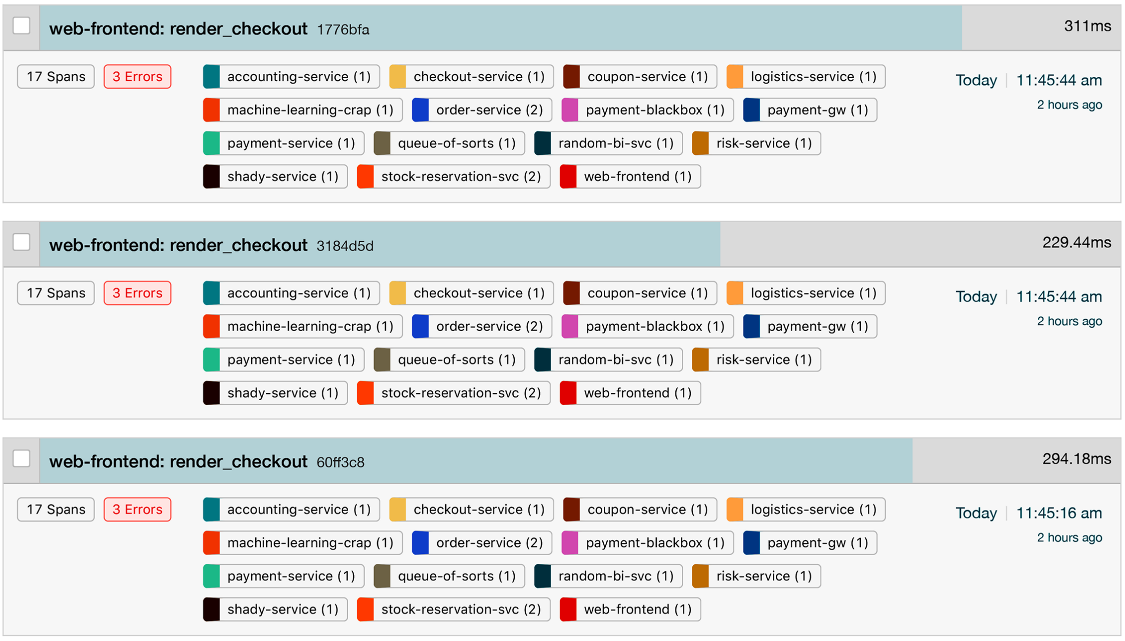
Latency, traffic, saturation, and errors are the Four Golden Signals [4]. If you can only measure four metrics of your user-facing system, focus on these four. They are great for alerting.

In this article we'll focus on one concrete signal—errors.

Alert Handler
Let's assume that an alert was configured for the *place_order* operation which has a service level objective (SLO) of 99.9 success rate. A typical way to measure this would be to query the tracing back end for spans that match a certain criteria. The keys *operation* and *component* are implicit on most tracing systems and represent the named span and the microservice itself, respectively. An expression such as *component: checkout_service && operation: place_order* represents the symptom and is where we want to measure customer pain. Different tools, open source and commercial, will usually provide different means to configure the alert itself. That's not in the scope of this article.

Adaptive Paging is an alert handler, and its architecture is broken down into three main components. The *transformer* is the actual alert handler, typically a webhook, and it's vendor specific. It's possible to have multiple alert handlers. The webhook receives alerts and converts them into symptoms. Then the symptom is passed to an evaluator, which implements the actual root-cause identification algorithm. The evaluator tries to determine the most probable root cause and generates a report. After the report is created it is made available to any reporter(s) which can deliver the page via different vendor-specific implementations or store debugging data to troubleshoot the alert handler itself.

Transformer
The transformer receives or collects vendor-specific exemplars and converts them into a vendor-agnostic data model that we called Symptoms. Exemplars are traces that should be representative of the symptoms that led to the alert being triggered. Some vendors can include exemplars as part of the alert payload. If they're not part of the payload, the transformer can query the tracing back end for exemplars that match the same criteria of the alert rule during the time of the incident.


```
web-frontend: render_checkout 1776bfa                                         311ms
17 Spans  3 Errors  accounting-service (1)  checkout-service (1)  coupon-service (1)  logistics-service (1)   Today | 11:45:44 am
                    machine-learning-crap (1)  order-service (2)  payment-blackbox (1)  payment-gw (1)          2 hours ago
                    payment-service (1)  queue-of-sorts (1)  random-bi-svc (1)  risk-service (1)
                    shady-service (1)  stock-reservation-svc (2)  web-frontend (1)

web-frontend: render_checkout 3184d5d                                         229.44ms
17 Spans  3 Errors  accounting-service (1)  checkout-service (1)  coupon-service (1)  logistics-service (1)   Today | 11:45:44 am
                    machine-learning-crap (1)  order-service (2)  payment-blackbox (1)  payment-gw (1)          2 hours ago
                    payment-service (1)  queue-of-sorts (1)  random-bi-svc (1)  risk-service (1)
                    shady-service (1)  stock-reservation-svc (2)  web-frontend (1)

web-frontend: render_checkout 60ff3c8                                         294.18ms
17 Spans  3 Errors  accounting-service (1)  checkout-service (1)  coupon-service (1)  logistics-service (1)   Today | 11:45:16 am
                    machine-learning-crap (1)  order-service (2)  payment-blackbox (1)  payment-gw (1)          2 hours ago
                    payment-service (1)  queue-of-sorts (1)  random-bi-svc (1)  risk-service (1)
                    shady-service (1)  stock-reservation-svc (2)  web-frontend (1)
```

Figure 5: Collection of traces (exemplars) that contain the failed operations (*error=true*)

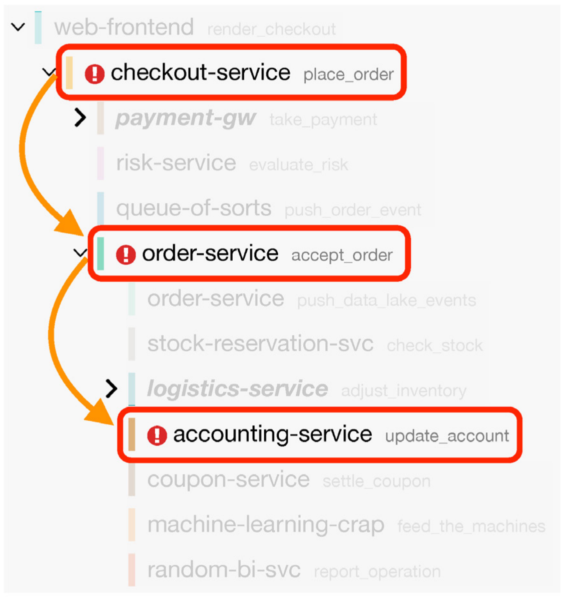

web-frontend render_checkout
checkout-service place_order
payment-gw take_payment
risk-service evaluate_risk
queue-of-sorts push_order_event
order-service accept_order
order-service push_data_lake_events
stock-reservation-svc check_stock
logistics-service adjust_inventory
accounting-service update_account
coupon-service settle_coupon
machine-learning-crap feed_the_machines
random-bi-svc report_operation

Figure 6: Probable root cause algorithm inspecting failed operations

Evaluator
The evaluation algorithm can have many different implementations. There can be different implementations for different signals—latency or errors, for example, or for any other known criteria for which a certain root-cause-identification algorithm performs better.

Example Errors Algorithm
The following example is one possible implementation to identify the probable root cause for errors. All exemplars (traces) are analyzed. Starting at the span that was defined as the signal source, each trace is inspected in a recursive way. For every child span, its tags and respective values are checked to decide which path to take.

In the example from Figure 6, none of the operations *take-payment*, *evaluate_risk*, or *push_order_event* were tagged as failed (**error=true**).

The *accept_order* operation in the order-service was tagged. The algorithm follows the path where **error=true**.

The same process is repeated. None of the operations of the *order-service*, *stock-reservation-svc*, *logistics-svc*, or the others which were triggered by *accept_order* were tagged with errors.

Only the *update_account* operation in the *accounting-service* was tagged as failed.

Without any child spans to continue the traversal, the *update_account* operation in the *accounting-service* is selected as the most probable cause of the errors.

After all exemplars are analyzed, a Report is generated.

Reporting
The Report generated by the evaluation algorithm contains information about the operation and microservice that is considered the most probable root cause. For reporters that page on-call engineers, the implementation needs to map the operation and/or service to the respective team or on-call escalation.

Putting It All Together
Going back to the original example, what happens if the Accounting Service has an outage and we're using Adaptive Paging? As you can guess, the team that operates the Accounting Service will get the single page triggered.

A similar situation would happen if any of the services involved in the "Place Order" operation breached its SLO, but the team that operates the probable root cause is the only one getting the paging alert—the one that will be able to actually do something about it—that is, no more page bombing.

Challenges
As mentioned before, the detection algorithm can adopt many different strategies. Zalando's current implementation uses a couple of heuristics that are easy to reason about.

Some of the things we had to work around when creating Adaptive Paging were:

- Multiple child spans tagged as errors: follow each path, attribute the probable cause a score. Analyze more exemplars and adjust the scores. Worst case scenario, page multiple probable causes. Paging two teams is still better than paging everyone.
- Missing instrumentation or circuit breaker open: either of these situations results in a premature evaluation of the probable root cause. We leveraged the semantic conventions to allow the caller to identify the callee, suggesting to the evaluator algorithm who to page, using the *peer.service=foo* and *span.kind=client* tag to suggest which service would be the target. This has the side effect of being a good incentive for teams to instrument their services.
- Mapping services to escalation: the service identified as probable root cause may not have a mapping to an on-call escalation. The evaluator keeps a stack of the probable causes and uses the one that is available and hopefully closest.

Finding probable causes due to latency is a challenge of its own. The strategy that we considered requires us to query the baselines for each operation and service combination, using that information to select which combination has a bigger variation at the time of the incident. This strategy can be a bit expensive, increasing the time to dispatch the paging alert.

Next Steps

Adaptive Paging was created with a multi-vendor reality in mind. Observability still has a ways to go, and some vendors are pushing the boundaries as we speak. Distributed tracing is still not a commodity, just like unit testing wasn't when it was initially introduced. No one would challenge the benefits of unit testing, and I believe no one will challenge the benefits of proper observability of distributed systems.

We've also started looking at some excellent work from LinkedIn—MonitorRank [5] from 2013, which fits nicely into Adaptive Paging; it's something we're considering as a possible improvement to the evaluator.

With Adaptive Paging we hope to contribute to improve the alerting situation, in particular paging alerts that burn out humans.

References

[1] Twitter thread on page bombing: https://twitter.com/mipsytipsy/status/1120911207903268864.

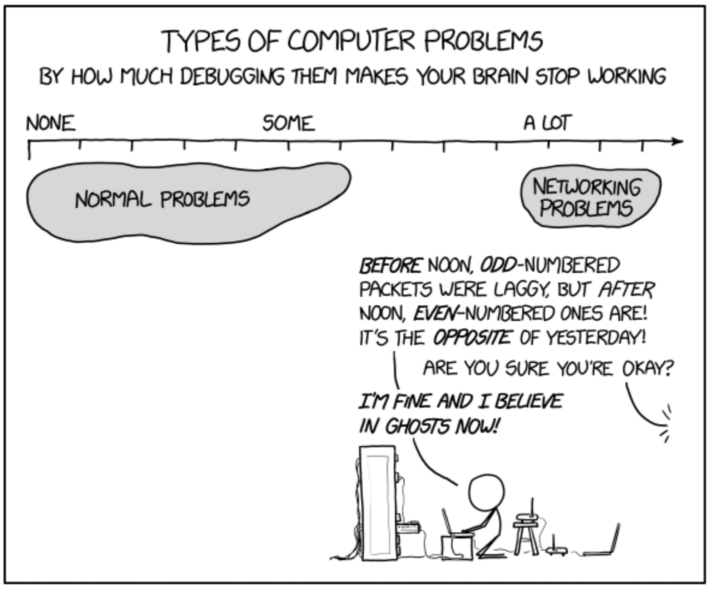
[2] B. H. Sigelman, L. A. Barroso, M. Burrows, P. Stephenson, M. Plakal, D. Beaver, S. Jaspan, C. Shan-bhag, "Dapper, a Large-Scale Distributed Systems Tracing Infrastructure": https://ai.google/research/pubs/pub36356.

[3] Jaeger: https://www.jaegertracing.io/.

[4] Four Golden Signals: https://landing.google.com/sre/sre-book/chapters/monitoring-distributed-systems/.

[5] M. Kim, R. Sumbaly, S. Shah, "Root Cause Detection in a Service-Oriented Architecture," SIGMETRICS '13: http://infolab.stanford.edu/~mykim/pub/SIGMETRICS13-Monitoring.pdf.

XKCD **xkcd.com**


TYPES OF COMPUTER PROBLEMS
BY HOW MUCH DEBUGGING THEM MAKES YOUR BRAIN STOP WORKING
NONE
SOME
A LOT
NORMAL PROBLEMS
NETWORKING PROBLEMS
BEFORE NOON, ODD-NUMBERED PACKETS WERE LAGGY, BUT AFTER NOON, EVEN-NUMBERED ONES ARE! IT'S THE OPPOSITE OF YESTERDAY!
ARE YOU SURE YOU'RE OKAY?
I'M FINE AND I BELIEVE IN GHOSTS NOW!

Getting Things Done

TODD PALINO

Todd Palino is a Senior Staff Engineer in Site Reliability at LinkedIn on the Capacity Engineering team, creating a framework for application capacity measurement, management, and change intelligence. Prior to that, he was responsible for architecture, day-to-day operations, and tools development for one of the largest Apache Kafka deployments. He is a frequent speaker at SREcon, as well as other venues, on the topic of SRE culture and best practices, and is the co-author of *Kafka: The Definitive Guide*, available from O'Reilly Media. Out of the office, you can find Todd out on the trails, training for the next marathon. tpalino@gmail.com

Two years ago, I found myself in a bad place, both at work and at home: overworking, ignoring my family, and angry all the time. It took months to understand the problem: I had no idea what work needed to be done. I could only focus on whatever was right in front of me, screaming for my attention. What I needed was a list of the work that I had committed to, that I trusted to be reviewed and complete, presented in a way that made it easy for me to pick the right work to do. I accomplished this with "Getting Things Done"—a process for handling work in a predictable and trusted way.

I lacked organization, which doesn't mean planning your day to the minute—as an SRE I live by the adage that no plan survives contact with the enemy. Not having organization meant I lacked the ability to respond appropriately to new work and ideas with a clear and creative mind. Martial artists practice "mind like water." Chefs have their "mise en place." For engineers, we have "inbox zero." You may call it a fantasy in an interrupt-driven world, but that only reinforces the need to have the planned work neatly maintained.

Most problems with inbox zero come from setting ourselves up to fail. We make a list of things to do today, leading to frustration when the inevitable interrupt happens and ruins our plans. We treat email inboxes as to-do lists, forcing us to continually re-read messages and decide each time what the next action is. Worse, we fail to fully catalog our work, both in the office and at home, and don't set aside the time needed for regular maintenance. When our partial attempts fail, we throw up our hands and declare organization to be an impossible task.

Enter GTD

There are many systems available for personal organization. For the last decade, I've used a system called "Getting Things Done" (GTD for short). Developed by David Allen, and documented in his book of the same name, the concepts have remained the same over the years, even as technology has changed. This is because it's not prescriptive regarding the tools that you use for organization. The process is described, with the characteristics that your trusted system must have, without placing bounds on implementation. It does not require specific software, or even any software at all. Last year, I was using a paper notebook.

You may not be sure what a trusted system is, but you're already using one: your calendar. We recognize that our brains are bad at remembering meetings, events, and the details for them, so we offload them into a calendar. Regardless of the calendar tool you use, when you get an invitation it goes into your calendar. You note whom you are meeting with, when and where the meeting is, and some details on the topic. Once you've done that your brain can let go of the information. This happens because you're consistent about using your calendar—you're checking it at appropriate times, or you trust that a notification will alert you just in time for a meeting.

This is the essence of a trusted system: a list of all of your commitments, which your brain trusts to be complete and regularly reviewed. We have to do this because brains do not organize information in a way that is conducive to getting work done efficiently. It believes that everything is important, all the time, so it continually cycles through your to-do list. It frequently drops items. It interrupts you randomly with information that you can't use at that time. In order to fix this, we need to make something other than our brain responsible for handling this information.

The tool is just one component—a way to store and present information. What makes it trusted is the process around how you use that tool. GTD's process is made up of five core steps:

- Capture
- Organize
- Clarify
- Reflect
- Engage

Capture
In order to organize work, you first need to collect the pieces of information that prompt us to create it. This is the essence of capture: create a habit of writing everything down. The goal is to only ever have a thought about something to do once—as soon as that happens, you write it down and it enters your trusted system by going into an inbox.

We have several kinds of inboxes. Email is just one type, and you probably have more than one account. Other inboxes include a notes app on your phone, your physical mailbox, and your pocket. It's just a place where you collect stuff that you need to do something with later. Know where all those inboxes are, but have the fewest possible. Most critical is to make sure you always have a way to make a note, paper or electronic, wherever you are.

As soon as you have an idea, write it down. This could be as trivial as "I'm getting low on milk" or as ambitious as "I'd really like to run a marathon." Treat this like brainstorming: don't filter. Capture all ideas, big or small, and only the idea: you don't need to figure out what the next step is, or even whether or not it's truly something that requires action. Capture is about getting it out of your head so you can continue with what you were doing with a clear head.

Organize
Before we discuss how to process everything we've captured, we should have somewhere to store our work. Like our calendar, this needs to be convenient to refer to wherever we are: in the office, running errands, or at home. Most will choose software for this, with far too many options to cover here. Two of my favorites that are tailored for GTD are OmniFocus and NirvanaHQ. Let's talk instead about what we're going to store in this system.

Actions are things that you can actually do. They are a single, discrete step: for example, a phone call. "Make a phone call and email a summary" is at least two actions. This is like a database transaction: we have to do the action all at once, or we roll back and start over. Our actions will not only have a clear statement of the work to do, they will also have a context. This is where you have to be, or what tool you need, in order to complete the action. Phone calls need a phone, so a good context is "Phone." Looking at a website requires "Internet." Locations can be contexts: there are things that can only be done at "Home," like organizing your spice drawer. People can be contexts, which is helpful for tracking delegated actions or agenda items for your next one-on-one meeting.

What's not needed are due dates or priorities. Priorities are a losing proposition, as you'll constantly waste time re-prioritizing work every time something new arrives. The context will help us filter down the number of actions available to us at any point in time, which will make it easy to see the important ones. As far as due dates are concerned, if something is time sensitive, think about whether or not it should be on your calendar instead. Doing this makes sure that the work gets completed before it is due.

The other concept is a project, defined as a desired result that requires more than one action. Examples might be "August vacation" or "Publish a book." It is a logical container for actions that accomplish a single goal. It's important to have these containers because a project is also a placeholder. When you complete the next action for a project, you need something to continue tracking that project and prompt you to define the next action.

Our organizational system comprises several lists:

- Next actions, preferably able to be organized by context
- Projects (a simple list is sufficient here because the actions will be on the previous list)
- A "Waiting For" list of all the actions we have delegated
- A "Someday/Maybe" list of the projects we might want to do later

Time-sensitive items should go on your calendar, which may be a separate trusted system, and reference items will be stored separately. This keeps your system reasonably sized, so you can carry it with you all the time. This is critical, because you have to be able to refer to it when you need to know what work you should be doing.

Clarify
Let's get back to all of the stuff that we captured. It's time to process it. Set aside the time on your calendar for this. I find that doing this any less than every weekday (except vacations) makes me anxious. I also triage my email inboxes throughout the day as I know there will be interrupts, like last-minute meeting requests. You may have certain inboxes that are processed less frequently, which is OK as long as you are consistent.

Clarify is a process of taking each thing in our inboxes and asking the question, "What is this?" The rule is that you will go through your inbox in order, one item at a time, and nothing goes back in. When you're done, your inboxes will be empty.

Select a single item or email and ask the first question: is this something that requires you to take an action? If not, it is one of three things:

1. **Reference.** Something you need to know, or refer back to later, such as a manual or other document. Reference items need to be stored, and there are many ways to do this: filing cabinets, flash drives, or bookshelves. Like inboxes, minimize the places you store reference and make items easy to find when you're looking for them.
2. **"Someday/Maybe."** Ideas you're not ready to commit to yet. For example, you might have "Run a marathon" or "Summit Mount Everest": maybe soon, but not today. This list of ideas will prompt you to think about them later on, to start when you're ready.
3. **Trash.** If it's not actionable, and not one of the above, throw it out. This might make you uncomfortable. Take that opportunity to evaluate your decision about whether or not it's actionable. If you can't throw it away, it probably represents something you need to do.

For actionable items, determine what the very next action is to move towards completion. Let's think about a couple examples from our day to day:

"Schedule one-on-one meeting." This is a single action: we need to send an invite for the meeting.

"Fix buffer overflow bug." This is not one action: we need to write the code to fix the bug, open and wait for a review, commit the fix, and deploy it. This is a project. We will add "Fix buffer overflow bug" to our "Projects" list. We also need the very next action to take, which is "Write the code to fix the bug."

Now ask how long the action will take to complete. If the answer is two minutes or less, do it right now. It will take less time than it will to track it. For scheduling a meeting, do that now because it will be quick.

Writing code is going to take longer. Actions like this are handled in two ways:

1. **Delegate.** Someone else will do it. If we ask a teammate to write the code, we'll add, "Alice—Code fix for overflow bug" to a "Waiting For" list. This tracks the action and who has it. When we review later, we might need to remind Alice about the work.
2. **Defer.** We will do it. Actions for a specific time can go on our calendar at the time it needs to be done. Otherwise, add it to our "Next Actions" list.

Reflect

It's not enough to put all of these projects and actions into a system, we also need to review that system with a consistent cadence. This is not the same as actually doing the work that we have defined—we're going to talk about that when we get to Engage. Reflect helps ensure that the system represents the totality of the work we need to do, as far as we are aware. This is the step that soothes the brain. When you understand, subconsciously, that anything in the system is going to have your eyes on it in some fixed and recurring time frame, only then will your brain be willing to let it go and trust the system. You know that you'll come back to it at the appropriate time.

How often do you need to reflect? It depends on what makes you comfortable, but a good start is to schedule a weekly review every Friday for two hours. This goes on your calendar because it's important to guard the time. Do not be afraid, or think it is selfish, to reserve time for yourself on your calendar. By doing so, you will make yourself more productive overall.

Routine and habit is the name of the game when it comes to GTD, and the weekly review is no different. Start with clearing your head: capture any thoughts in your head that are bouncing around, and process your inboxes to zero. Then you're going to move into reviewing your entire trusted system to make sure everything is current:

- Look at "Next Actions" and check completed actions or capture new ones.
- Review last week's calendar to make sure you captured action items.
- Review next week's calendar to surface actions to prepare for it.
- Check your "Waiting For" list, sending any needed reminders.
- Review your "Projects" list, making sure each has a next action.
- Review your "Someday/Maybe" list, and pull anything to start into the "Projects" list.

After you finish the weekly review, you're going to feel like you really have your life together. This is one of the reasons I like to do it on Friday afternoon: it lets me go into the weekend knowing that I'm fully organized, and I can set work behind me and be present with family and friends.

Engage

So far we've talked about organizing things, not actually doing them, and there's a good reason for that. When your work is well organized, it's easy to select the right thing to do at any point in time and get it done. You're going to be working from your "Next Actions" list, because this represents all of the things you can do right now without waiting for something else. With the previous four steps in place, we are comfortable that the next actions list represents the totality of the work that we are aware of. We can quickly narrow down what we can do right now on the "Next Actions" list by four criteria:

- Context: Filter out actions that don't match the contexts available to you.
- Time: If you have 15 minutes available, you can't do an action that will take longer.
- Energy: At the end of the day, you might only have the brain-power to read an article.
- Priority: With actions filtered down, it's easy to pick out the highest value one.

What happens when you're interrupted with an alert or some other interruption? First, don't get sucked into automatically doing work as it appears. Knowing your planned work helps with the decision on whether the new item is a higher priority. If it is, set aside your planned work and focus on the new work. Your trusted system will be there when you finish, and you don't need to worry about keeping track of where you were. Just pick up the next action that fits based on the four criteria.

Organizing your work will result in fewer of these interruptions, as well. Many of them exist because we didn't know what our commitments were. We forgot about that bug fix we meant to do. We buried an email that asked us to review a document by a certain date. Prioritizing proactive work will reduce the amount of reactive work required.

Next Actions

Organization is a project, and here are some next actions you can take to free up your brain to do what it's good at: being creative and solving problems.

1. Borrow or purchase a copy of *Getting Things Done* by David Allen.
2. Read the book.
3. Select GTD software (or other tool) to implement your system.
4. Schedule time with yourself for your first pass through Clarify and Organize.
5. Schedule time with yourself for a weekly Reflect session.
6. Schedule time with yourself for a daily Clarify session.

What I have presented here is an overview: there is more to be gained from reading through David Allen's book. You'll gain a deeper understanding of how to work with the GTD concepts as well as an introduction to other topics, such as horizons of focus—how to work with long-term planning and frame the question of what you want to be when you grow up.

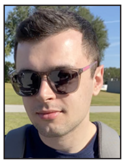
The first time you work through processing inboxes, it's going to take a lot of time. You'll need to look at how many emails, and other pieces of paper, you have and make a decision about how much time you need. Don't shortchange yourself—time spent here will return to you 10 times over. Break it down into multiple sessions if needed, but don't leave too much time between them.

Finally, remember that organization is a habit that you need to build. Getting Things Done provides a structured process for managing our work, but it only works if you follow it consistently. It will take time to remember to capture every idea, and you will need to be diligent about guarding your time for both Clarify and Reflect at first. The feeling that you get after clearing out your inboxes and reviewing your trusted system will ensure that the habits, once established, will be hard to break.

It's an SLO World
What Theme Parks Can Teach Us about User-First Reliability

JAIME WOO AND EMIL STOLARSKY

Jaime Woo is an award-nominated writer and has been published in the *Globe and Mail*, *Financial Post*, *Hazlitt*, and *The Advocate*. He spent three years as a molecular biologist before working at DigitalOcean, Riot, and Shopify, where he launched the engineering communications function. He's spoken at SXSW, IA Summit, SREcon Americas, and SREcon EMEA, and was a guest lecturer at the University of Toronto's Rotman School of Business. He is co-founder of Incident Labs and is co-authoring the forthcoming book *SRE for Mere Mortals*. jaime@incidentlabs.io



Emil Stolarsky is a Site Reliability Engineer who previously worked on caching, performance, and disaster recovery at Shopify and the internal Kubernetes platform at DigitalOcean. He has spoken at Strange Loop, Velocity, and RailsConf, was a program co-chair for SREcon19 EMEA, and is a program co-chair for SREcon20 Americas West. He has guested on the podcasts InfoQ and Software Engineering Daily, and contributed a chapter to the O'Reilly book *Seeking SRE*. He is co-founder of Incident Labs and is co-authoring the forthcoming book *SRE for Mere Mortals*. emil@incidentlabs.io

In an always-on world, predictable reliability is paramount. Service level indicators (SLIs) and objectives (SLOs) are cornerstones in site reliability engineering (SRE) for purposeful reliability. SLIs are chosen measurements that act as signals for achieving your reliability goals; SLOs are the targets for SLIs. User-first SLIs and SLOs are the gold standard, and we use the concept of theme parks, those paragons of complex systems optimizing for user happiness, to demonstrate examples of strong SLIs and SLOs in contrast to useless ones.

The massive, iconic theme parks of Orlando, Florida, are impressive for children—the rides, character actors, and sights synthesize into magical, larger-than-life playgrounds. It was surprising then to realize how much more impressive the spaces become when revisited as adults. The infrastructure that manages hundreds of thousands of visitors daily, the attention to detail across the "lands"—even in places where people might not immediately notice—evoke awe and appreciation for the levels of planning and effort.

The lessons for site reliability engineers from theme parks are not immediately obvious, until you realize that SLOs are rooted in asking what level of service must be provided to keep users happy. And where else could you glean lessons about how to engineer for happiness than at the so-called happiest place(s) on earth?

Useful SLIs

Let's begin with how to find a useful SLI. A useful SLI must contain the following four parameters:

- Relate to the experience and/or satisfaction of your user
- Use a measurable quantity related to your service level
- Be as specific as it can be
- Provide enough information to be actionable

Similar to the massive infrastructure that is behind what appears as simple user-facing experiences, underneath the colorful, playful facades of theme parks are subterranean levels where workers manage the infrastructural and logistical components of the park, including electrical operations, transporting character actors, waste removal, deliveries, and food service [1].

Visitors rarely think (or even know) about these hidden parts—and that's the preference of theme parks so as not to ruin the illusion. The only thing that matters is the experience visitors paid to have, and everything that is out of view exists only in support of that experience.

Take waste removal: should trash begin to pile up around the park, visitors would complain about seeing garbage on the park grounds, rather than, say, faulty waste removal mechanisms 20 feet below them. And, theoretically, those mechanisms could break and guests wouldn't notice whether staff cleared the paths of trash often enough. So the amount of garbage on the floor is a stronger SLI than waste removal machinery uptime.

That doesn't mean ignore everything internal: instead, we acknowledge that something can be important yet not necessarily urgent. That ambiguity around urgency highlights the disadvantage in using such metrics as guidance for reliability: because it's subjective, it's more difficult to gauge reasonable boundaries around allowable downtime and, therefore, to create meaningful error budgets to justify any downtime. Anything that users interact with directly affects their ability to do what they need to do, and thus prioritizing work is clearer.

For example, take a database service with multiple replicas. It might be tempting to use uptime as an SLI, but there are many scenarios where uptime gets dinged but customers don't feel the impact. For instance, if a single replica goes down, traffic won't be affected. Instead, a more effective SLI would be to track read-query success rates, which are necessary for customer requests to be successful.

From SLIs to SLOs

Upon determining SLIs, you have to assess the right target SLOs. From our theme park example, we've figured out that guests would be unhappy with trash everywhere. Now, we want to know what their threshold is, based upon their needs and expectations.

With small piles of garbage everywhere, the park technically remains operational, but it would be a poor experience for guests, potentially discouraging them from returning or even asking for refunds. On the other hand, ensuring no piece of trash stays on the ground for longer than a few minutes would be an excessive waste of resources. How then to choose the right level?

Luckily, engineers need not—and should not—do this alone. Different business units across the organization will have their own insights into users, and when site reliability engineers work with teams like support, engineering, and product and bring those insights together, you're likelier to have meaningful SLOs. At Disney World, you're unlikely to see trash on the ground for longer than 15 minutes, as that's the interval, in crucial locations, at which trash in bins get sucked into an underground automatic vacuum collection (AVAC) system and transported away.

With our example SLI of read-query success rates, after discussing with other business units, we may learn that users typically notice degradation in the service when fewer than 95% of read-queries succeed over a period of 30 minutes. Waking up engineers the moment any read-query fails would be premature, but we could set a slightly more stringent internal SLO that once read-queries drop below a 97% success rate, alerts get sent out.

What Is the Experience?

With the need to be user-focused firmly established, we can move from what users see to what users experience. The distinction between the two is that one measures what users interact with, and then the second looks at those interactions and translates them into their meaning. In the field of UX, they understand the distinction: "While you cannot directly *design* a person's experience of a product, you *can* take steps to ensure that their experience is a positive one by employing a user-centered design process," writes Matt Rintoul, experience design director of creative agency Say Yeah! [2].

At this point, we should make a vital distinction: who your users are matters. For this article, we focus on human users rather than programs that use your service (although users can be both, depending on which part of the service each touches).

Thinking about how different users interact with your system is a useful exercise. The response times from programs are more reliable and faster than with humans. You can also tell a program to attempt a request again in five minutes. Unlike machines, humans perceive things relatively, something we'll return to later in this piece. This matters because treating humans as rational actors, as economists do, can simplify things, but you need to be careful it doesn't oversimplify. Context matters.

What Constitutes a Satisfactory Experience?

Rarely is a service entirely down. Instead, individual components may lag or fail, and even with some parts of the experience deprecated there may be no change in a user's core experience. At a theme park, the food stands, for example, could be out of service, and the park could still run. If the restrooms all failed, however, it'd be a different story.

For a technical example, on a video-streaming service, there are many components to the total experience, from searching content, user-curated lists, viewing history, and playing content. Each component can be mapped to see if it is running or not, and this matters because the components are weighted related to user satisfaction: if customers can still continue watching a film they were in the middle of then they will be happy even if they cannot amend their list of media to watch.

Specificity matters because when outages occur, or decisions around what work should be prioritized, you make best use of your limited resources by understanding which parts of the service matter most to customers. You can then also manage the number of things being tracked in dashboards to prevent information overload. An engineer needs to weigh the tradeoff of adding another SLI to monitor against the level of dissatisfaction users will have if it goes down.

Timing Matters

Just as components of your service are relative, with differing weights, this is also true for time: not every minute is the same. If your users don't notice an outage, should it count toward your error budget? At theme parks, for instance, electronic gates require fingerprint identification for entry. If this system went down an hour before the park closed, while that’s not ideal there’s also the nontrivial question of who was impacted?

This isn’t permission to ignore outages that happen during the off hours. You still want to know how often your service is going down, which provides a better way to understand the behavior of your system. But does your service truly need to be up 24/7? Are there periods when the service is lightly used? It isn’t zero impact, but it has less impact, so do your metrics reflect this? Importantly, is a low-impact event worth the human capital of waking a team at three in the morning?

As an example, a food delivery service that solely works with restaurants on the East Coast of the United States: most restaurants do not operate between two to six in the morning, and perhaps the service has data showing that orders drop off after 10 p.m. and only revive at 10 a.m. for lunch orders. An SRE team could decide that alerting overnight for low-severity incidents isn’t worth sleep-deprived and grumpy engineers and instead send pages in the morning.

Users Have a Multitude of Experiences

Rarely are users a homogeneous monolith. Instead, they are heterogeneous, each with their own (albeit, potentially overlapping) needs. In UX, the practice of creating personas acknowledges that users have different perspectives and rationales. When considering SLIs and SLOs, we should avoid blanket aggregation of users for the sake of simplicity.

Returning to the theme parks of Orlando, think about the different types of visitors: parents and guardians, children, aunts, uncles, grandparents, and adults without children. They speak different languages. They have different accessibility needs. They have different cultural perspectives. As a result, theme parks provide experiences to cater to the wide range of needs and expectations.

An example was the introduction of single-rider lines: rides often seat visitors in pairs and therefore can have unused capacity when groups have an odd number of people or for solo visitors. Worse, solo visitors would wait as long as large groups, even as they could see empty seats on the ride. By creating a line just for individual riders who don’t mind sharing with strangers, the excess capacity can be used up—providing a quicker queueing experience for all guests.

Users of technical systems are just as varied. They can come from different geographic regions, be of different sizes, vary in their frequency of use, and so on. And aggregating them is just as pernicious. An example is when a company has their datacenter in North America, where the majority of their customers are. If the data is aggregated, a customer located in Eurasia facing subpar performance might not trigger an alert: the user may become unhappy, even if all SLOs appear to be met.

User Perception Matters

Unlike machines, humans perceive interactions based on their past experiences and attempt to create context based on what has happened: a machine might make several attempts to connect without those attempts creating any kind of storyline. This is less true for humans, where they build theories based on patterns, and it is at our own peril to ignore this fact.

We cannot, obviously, measure how users feel at every moment because it is intrusive and expensive. We also do not want to rely on users venting their frustration at customer support or online on Twitter either, because then it’s too late. But we can start thinking about user perception as a factor in our SLOs and acknowledge that it plays a role if we are to be truly user-focused.

Perception is by definition subjective, sometimes in counterintuitive ways. An illustration comes from a phenomenon called paradoxical heat: when a person holds a warm pipe in their left hand and a cool pipe in their right hand, they sense painful heat, even if neither pipe individually feels unbearable. We are unaware of a directly analogous phenomenon for SRE, but a similar idea might be having two minutes of downtime, followed by two minutes of availability, followed by another two minutes of downtime.

This won’t feel like four minutes of outage: anyone who has experienced spotty WiFi coverage will understand the oddly intense anguish that comes from intermittent connectivity. It can feel worse than not having Internet access at all, because it robs you of your sense of control over the situation: should you keep trying or do something else? Not knowing whether the next outage will be in a minute or not at all can be very frustrating. So we can’t just look at the raw data itself but have to also think about how that data represents experiences. Four one-minute outages alternating over an eight-minute period may feel worse than a continuous four-minute outage.

Perception also plays a role in the least interesting part of visiting a theme park: waiting in line for a ride. However, huge investments have been made to create engaging and sometimes interactive experiences during the queue to make the experience feel less painful. Before a Harry Potter-themed ride, visitors roam an immaculate set modeled after Hogwarts, the fictional wizarding school, and the immersion makes the time seem to go by faster.

Theme parks also post estimated wait times so that visitors feel a sense of control about whether or not to join the queue—and these times are padded so that guests feel delight at “saving” time. Isolating wait time provides some information, but if you have set up a standard and even sticking to it leads to unpredictable outcomes, then you must realize that you need more information to guide your decisions.

How do you learn about these expectations? You can look at user-behavior data, such as when customers drop off, and try to figure out a trend. Or you can ask them directly through surveys and interviews. But it’s important to think about when is the right time and place to ask them. Asking after a major outage will yield different answers than after a period of calm, and asking them before their issue is resolved is different from asking afterwards.

You will also want to pair up with someone who understands how to craft useful survey questions: for example, you do not want to create leading, ambiguous, or unclear questions, and you want to use a Likert scale. Poorly designed survey questions lead to low quality data, and sometimes people can assume that surveys are the problem, but that’s blaming the tool rather than the person wielding it: more than likely it is how surveys are created and conducted that are the problem.

Benefits

We all have limited resources, especially time. When we choose the most meaningful, user-focused SLIs and SLOs, we make the most of those resources. You’re prioritizing for the experience your users want and creating the boundaries for services. If something goes down, but it doesn’t impact user experience, it’s still important, but it isn’t necessarily urgent. Just because we can do something doesn’t mean we should. We can wake people up in the middle of the night to manage an incident, but are we alerting for the right things? What matters and what doesn’t?

There is a broader benefit: the third age of SRE is upon us, and it is one that posits that reliability is cross-functional, something that not just developers and technical project managers need to think of, but also accountants, lawyers, and customer support teams.

Yet this isn’t a one-way street. Just as everyone should have a reliability mindset, we must remember why reliability matters. It’s not just done for its own sake (and actually can be costly, a detriment to feature velocity, and cause for burnout) but because customer experience matters and customers demand reliability. Reliability that doesn’t include a user-focus is only tackling part of the problem, and when it becomes more developer-focused than customer-focused it becomes about ego. So everyone must have a user-oriented mindset.

Such a shift can be frightening because users can seem subjective, but, unless our only users are machines, that’s how it goes. What we can do is approach it differently, with wonder and excitement. How can we delight our users the way theme parks spark joy for visitors? Our favorite example of thinking about users: at the Disney World parks, designers created different floor textures for each land, so that even your feet know when you’re moving into a new experience. It may be at a level beyond what we need, but we can afford to walk a few steps in the right direction.

References

[1] https://en.wikipedia.org/wiki/Disney_utilidor_system.

[2] M. Rintoul, “User Experience Is a Feeling,” UX Matters, October 2014: https://www.uxmatters.com/mt/archives/2014/10/user-experience-is-a-feeling.php.

Interview with Mary Ann Horton

RIK FARROW

Mary Ann Horton has been a UNIX developer and sysadmin since 1977. She contributed to Berkeley UNIX, creating the first email attachments and enhancing vi. Her PhD dissertation at Berkeley led to IDE editors that check your program for errors. While at Bell Labs, she led the UUCP Mapping Project and brought .com domains to UUCP email. She led the growth of Usenet, an early social media network, in the early 1980s. Her EMS email system allowed email addressing by database query. As a transgender activist in the 1990s, she convinced Lucent Technologies to become the first large company to include gender identity and expression language in its EEO nondiscrimination policy, and later to cover transgender hormones and surgery in its health insurance. At San Diego Gas & Electric, she designed SCADA control systems to make the power grid more reliable, secure, and compliant with regulations.
www.maryannhorton.com
mah@mhorton.net

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

I met Mary Ann Horton at USENIX ATC '19 in Seattle. I didn't know who she was, but somehow discovered that she worked on the control systems for the grid in the San Diego area, and we exchanged email addresses so we could continue the conversation. Later, I read her Wikipedia page [1] and learned much more about her.

Rik Farrow: You have been working with UNIX since its earliest days.

Mary Ann Horton: I fell in love with UNIX earning my master's degree at Wisconsin in 1977, but my big break came in 1978 when I transferred to Berkeley for my PhD. We got a VAX, initially with VMS, but quickly changed to UNIX 32/V. There were many amazing grad students contributing tools to BSD, and it was a treat to get to be part of this effort. It seemed like about half the code was written by Bill Joy, including `vi`. I got to enhance `vi`, nurture it, and port it to all sorts of UNIX clones. Eventually I replaced `termcap` with `terminfo` and wrote a new improved `curses` library so other programs could work as well as `vi` on slow terminals.

My doctoral dissertation was a language editor, which meant you were editing a program tree but it seemed like a text editor. The editor had parsed your program, so it could show you your syntax errors, and even some semantic errors. It was horribly slow on the VAX, but the technology was used later in IDEs like Visual Studio and Eclipse.

I needed to email binary files, but UNIX email only supported plain text. In 1980 I wrote a dumb little program called `uuencode` to embed binaries into text email, and `uudecode` to extract them. In 1985, Lotus and Microsoft decided that `uuencode` was the existing standard format for attachments and used it in their PC email systems.

RF: What was it like to work at Bell Labs in the Midwest? We often hear about the more famous branch of the Labs in New Jersey, and how researchers there appeared to have a lot of freedom to develop many of the things we take for granted today.

MAH: I was a summer student at Holmdel, New Jersey, in 1979. I loved Holmdel but hated living in New Jersey. I wanted to do UNIX work at Bell Labs with the official Research Center 127 folks (Ken, Dennis, etc.), but policy was that all research was only in New Jersey. When Dale Dejager recruited me for the new Exploratory Software Group (ESG) in Columbus, I jumped at the opportunity. My wife's family lived in Ohio, and she wanted to move there. I started in 1981 after I finished my PhD work at Berkeley.

Bell Labs was the R&D arm of AT&T. The Columbus Works (CB) was a Western Electric factory, with a Bell Labs office building attached to the front. Money flowed freely, and there was plenty of computer equipment. In the days of expensive long distance calls, nobody cared about the phone bills we ran up when our UUCP network dialed another computer to exchange email and Netnews. CB was dwarfed by the larger labs in New Jersey and the Chicago area, but we all respected one another and shared a love of technology, especially UNIX.

The ESG was kind of like minor league research. It was a spin-off from the Operating System Group, where Dale and a group of UNIX experts had created an enhanced "Columbus UNIX" for the needs of telco Operations Support Systems developed in Columbus. "CB-UNIX" supported

IPC, shared memory, and semaphores, as well as some Berkeley enhancements like `vi`. CB-UNIX ran on the PDP-11/70 only, especially `cbosg`, the OSG's main home machine and email hub.

When I arrived I was presented with a shiny new VAX 11/750, where I promptly installed 4BSD on the new `cbosgd`. I set it up with email and Netnews, and it became the main connection into Bell Labs CB.

By 1983, Bell Labs's UNIX Support Group decided to support the CB-UNIX features and add `vi`, so the OSG and ESG were disbanded, and we all moved into development for new products. In 1987, we created a Gateway Group to formally support email and Usenet gateways, where I spent the next several years.

RF: By this day and age, people who know what Netnews was are disappearing. Netnews was, in itself, a precursor to the Web. Just supporting Netnews was difficult, and I understand you had both a programming role and a social one in developing Netnews.

MAH: Usenet was one of the first social media networks, carrying the Netnews traffic. I first heard about it in 1980 at the Delaware USENIX Conference, when Steve Daniel and a crew from Duke and UNC gave a paper about it. You could post a message on a newsgroup, and within a day it would be visible on Usenet hosts all over the country! This was awesome, and I brought it to Berkeley as soon as the conference software tape came out.

In those days nearly all UNIX networking was via dial-up UUCP links, and long distance phone calls were expensive. Universities like Berkeley had strict policies not to let their computers make long distance calls. Tom Truscott at Duke had spent a summer with the Research group at Bell Labs, so Research's UNIX system called Duke's every night to pick up email, and they also called Berkeley's `ucbvax` system nightly. Bell Labs didn't mind a phone bill, so when we added Netnews links through Research they picked up the expense.

The "A News" software from Duke/UNC was intended for low volume, but the traffic grew quickly as more and more people got onto the Net. All the traffic went into one directory, and users saw everything in the order received, a UNIX posting, a recipe, a car for sale, a response to the UNIX posting. It seemed really disorganized. I had the idea that Netnews should be like email, with header lines.

One day a high school student named Matt Glickman walked into my office at Berkeley looking for a project, and I suggested "B News." I designed it and he got it coded over his spring break. B News organized the new postings by newsgroup, and expired old news after a couple of weeks. The newsgroup net.unix carried discussions about the UNIX system: net.cooks was for recipes, and we had net.jokes, net.autos, net.politics, net.jobs, and on and on.

There actually were earlier social media: ARPANET mailing lists and BBS systems. The ARPANET had a Telecom list, UNIX-Wizards, Human-Nets for Human Factors, and SF-Lovers for Science Fiction. These were busy lists with lots of interesting traffic, but only available on the ARPANET. I rigged up a gateway at Berkeley to post the traffic to Usenet, with a new hierarchy "fa.*" for "From ARPA" to make it easier for sysadmins to choose whether they wanted it.

By 1981 I found myself helping more and more universities and companies get onto Usenet. They needed to find a kind sysadmin who would give them a dial-up connection and a news feed, so I came up with a "pay it forward" rule where, if someone gave you a connection, you should be willing to give at least two more systems connections in return. That kind of spread the load around, and Usenet kept growing.

When you logged into UNIX, it would tell you "You have new mail," and we added an option to B News for your .profile so it would also say "There is news." The joke became "There is always news." Catching up on Netnews got to be time-consuming, and you could get sucked into the vortex, just like Facebook today.

Netnews was mostly for fun, but people needed email for their work. In those days you had to give your email directions: duke!unc!research!ucbvax!mark, so people needed a map of UUCP connections just to send email. I handed out Usenet logical maps at the USENIX conferences in 1982 and 1983, and people snapped them up to use to route their email. There was constant confusion between Usenet, which carried Netnews, and UUCP, which carried email and was much larger and better connected. By 1984 I gave up distributing a logical map, because it was too branchy. I handed out a geographic map, and Bill and Karen Shannon put out an eight-page logical map [2] at the 1984 USENIX conferences. After that it was just too big to draw a picture.

Usenet distributed Netnews with a "flood algorithm" where a node sent all new news to each neighboring node, which checked each article to see if it already had it. If not, it saved the news and sent it along to its other connections. The "Path" header showed how the article got there, so any node on the path could be skipped. That way everything worked its way around the Net, with some redundancy. But growth made it unwieldy, so in 1983 I set up a "Usenet Backbone" that would carry the news around the world in an organized fashion, then send it out for local distribution. Gene Spafford (then at Georgia Tech) didn't see Atlanta on the Backbone, so he got involved about 1984. He realized the sysadmins on the Backbone had power to run the Net, so he set up the "Backbone Cabal" mailing list as a political decision-making group. Backbone maps were still manageable, so we put out a few of those from 1983 to 1986.

One of the biggest Backbone hosts was `decvax`, run by Armando Stettner and Bill Shannon of DEC. They called so many impoverished universities that, at one conference, Armando bragged about a quarter-million dollar phone bill just for `decvax`. There was even a rumor going around that Usenet was really a scheme of AT&T to generate revenue by running up phone bills! The truth was just the opposite: sysadmins ran up their phone bills quietly so their bosses wouldn't notice, and AT&T corporate had no clue any of this was going on. Bell Labs, however, had the largest presence on Usenet, brought in through gateway machines like `ihnp4` and `harpo`, with friendly sysadmins like Gary Murakami and Brian Redman.

RF: In the '80s, there were many email standards. I see you worked in that area as well.

MAH: There were too many incompatible email standards. The ARPANET was great: user@host got your mail there. There were also CSNET, BITNET, FIDONET, and a hot mess called X.400. AT&T used UUCP, but internally it was pretty well connected. All AT&T systems were registered with Network Action Central, so host!user worked for any AT&T internal email. But the rest of UUCP was ad hoc, so everyone had to route their own email, and an address like research!greg@Berkeley was ambiguous.

As the Net grew, several people offered to create an email map but disappeared under a mountain of UUCP system files and were never heard from again. That all changed when Internet domains came along. I was an early advocate, writing a paper "What the Heck Is a Domain" extolling their virtues. I thought they could be used for UUCP email, so at the 1984 Washington USENIX Conference I got a BoF session together to plan the map. We created the UUCP Mapping Project, all volunteer based, to post and update UUCP connection information to comp.mail.maps on Usenet.

Peter Honeyman and Steve Bellovin wrote `pathalias`, which converted the map information to a localized email routing database. I helped a high school student, Adam Buchsbaum, write the `smail` program to send email using the database, and we set up the .UUCP top level domain to go with .ARPA. It worked great for us, but the ARPANET didn't recognize any other domains, so they had to resort to addresses like mark%cbosgd.UUCP@Berkeley. ARPANET addresses were only allowed one @ sign, so a second @ had to be hidden as a %.

I represented UUCP at a January 1986 meeting at SRI, along with Craig Partridge from CSNET and Dan Oberst from BITNET. We wanted official ARPA recognition of our domains. Ken Harrenstien from ARPA convinced us that all the world should be under six domains: .COM, .EDU, .ORG, .GOV, .MIL, and .NET. Steve Kille from the UK asked for a special clause for other countries to use their two-letter country code, but didn't expect it to be used much. We were all authorized to share registration of domains in the big six through Jon Postel at ISI. I had to take the UUCP Project to the next level, so we set up Stargate Information Systems as one of the first domain name registries. The first domain I registered was stargate.com, and the second was att.com. We got it working with `smail` and brought lots of UUCP-only companies and universities into the .com and .edu spaces. The Stargate side of it was Lauren Weinstein's project.

In 1992, Bell Labs had two competing email systems: `smail` understood domain addresses and worked with `sendmail` on Suns, but much of the labs had an email system called POST. A team maintained a database of AT&T staff, including name, location, title, and their host!user email address. Their `post` front end to `mailx` allowed you to send email to people by name, so `post john.bagley` would let you compose an email, just like `mailx`, and deliver it by looking up his UUCP email address and handing it to an email back end called `upas`. You could send to groups with queries like `org=4526`, which went to an entire department, or `tl=sup/loc=cb`, which went to all the supervisors in Columbus.

I started fiddling with `sendmail` and `post` and integrated domains with the `post` lookups. I called the system EMS, and it was deemed useful enough to form an email team to support it. Now I could put mark.r.horton@att.com on my business cards. One day my email stopped working. When I dug into it, I discovered AT&T had hired another Mark R. Horton! When AT&T spun off Bell Labs and Western Electric into Lucent Technologies, the POST team added a "handle" field for email, first come, first served, so I became mark@lucent.com.

In 2000, Lucent spun off Avaya and Agere, and I went to Avaya to manage their email and POST directory team. That lasted a year, then the .COM bubble burst and I took a package from Avaya.

RF: You wrote in your Wikipedia page [1] that you got Lucent to provide support for your gradual transition, starting with cross-dressing. I can't imagine that that was easy.

MAH: Most transgender people know they're different from an early age, but I took a long time. I first got interested in women's clothes at age 10, but didn't really fully cross-dress until 1988 at age 32. My first wife Karen divorced me over it, but my second wife Beth was supportive. We kept it a deep secret for years, but I started to come out in 1996 when Lucent's gay/lesbian/bi group, EQUAL!, added transgender to their mission.

I went to an EQUAL! conference in Denver, the only trans person there, and we educated each other. I learned how important it was for gay and lesbian people to have the freedom to come out in the workplace, empowered by the words "sexual orientation" in the EEO nondiscrimination policy. They didn't have to spend energy hiding part of themselves, so they were happier and more productive at work.

Unfortunately, those words didn't protect me. I asked, through channels, if HR might add transgender language to the EEO policy. Later, the question came back, "If we were to add transgender language, how would we do it to be as inclusive as possible?" Opportunity was knocking!

I didn't know the right language, but I had connections in the trans activist community. A trans attorney in Washington, DC suggested "gender identity, gender and sexual characteristics, and gender expression." What a mouthful! It turned out "sexual characteristics" didn't fly with HR, so we condensed it to "gender identity, characteristics, or expression." Rich McGinn, the Lucent CEO, signed it in 1997, the first large company to officially include transgender people in their EEO policy!

By this time I had a life as Mark and another life as Mary Ann, and I was allowed to come to work occasionally as Mary Ann. I was also an activist for trans rights in other places, and the other trans activists were all transsexuals who had transitioned long ago. They all said corporate America could not handle a part-time cross-dresser in the workplace, but I was already doing it!

Somebody was afraid I might want to use the restroom, and a secret meeting was held by HR, Corporate Security, EQUAL!, Medical, and my boss (but not me). They decided I should use the single occupancy restrooms in Medical, a quarter mile from my office. A bathroom break took 15 minutes, but I was so happy to be able to go to work as Mary Ann that I accepted it.

Through EQUAL! I was able to get other Lucent policies improved. When people realized that the world didn't end if I used the ladies room, I persuaded HR to use the principle of least astonishment, so that transgender people would use the restroom for the gender they are presenting.

I knew other transgender Lucent employees who could not get their medical care covered. I worked with HR to cover the hormones and surgery, and in early 2000 someone got her surgery covered. I still hadn't transitioned, so I didn't personally need the medical coverage.

I didn't transition until late 2001, after I'd been spun off to Avaya and taken their early retirement package. Beth supported my transition, but she didn't want to be married to a woman. We broke up and I went looking for a job as Mary Ann. The market was flooded with great people after the .com bubble burst, so it took 11 months to get a UNIX job with Bank One. I used that time to become legally and medically female.

After success with Lucent, I was an activist getting other companies to add Gender Identity and Expression to their EEO policies. Apple followed quickly, then Chase and IBM. In 2002 I (with other activists) convinced the Human Rights Campaign to include points for transgender nondiscrimination language, and HR departments began to quickly add it.

In 2001, inspired by Lynn Conway's "back of the envelope" calculation asserting that it would be cheap to fully cover transgender surgeries, because so few people have them, I surveyed the surgeons. An astounding 75% responded, and I was able to get a pretty good estimate of the total cost to cover transgender health benefits, which came out to about .004% of total health premium costs, basically the change in your couch. I gave workshops at the Out & Equal LGBT workplace conferences, and other companies began to add the coverage. Now it's standard coverage in large companies, at zero added cost.

RF: What did you do after leaving Bank One?

MAH: In 2007 the stars were aligned and I got to come back to California. I worked for San Diego Gas & Electric supporting the SCADA system that runs the transmission grid. After the Northeast power blackout in 2003, there was a lot of concern about reliability and cybersecurity, and I wound up leading the effort to keep the hackers out. There are a lot of NERC CIP regulations [3] about this, so not only did we have to keep the system secure, we had to provide daily evidence we were doing so. I wound up building automated tools to collect the compliance evidence.

I retired in 2018. Now I'm enjoying living in paradise and writing a memoir.

References
[1] Mary Ann Horton: https://en.wikipedia.org/wiki/Mary_Ann_Horton.

[2] UUCP/Usenet maps: stargatemuseum.org/maps.

[3] NERC CIP: https://www.ispartnersllc.com/blog/nerc-cip-standards-overview/.

Save the Dates!

usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Sixteenth Symposium on Usable Privacy and Security

Co-located with USENIX Security ’20
August 9–11, 2020 • Boston, MA, USA

The Sixteenth Symposium on Usable Privacy and Security (SOUPS 2020) will bring together an interdisciplinary group of researchers and practitioners in human computer interaction, security, and privacy. The program will feature technical papers, including replication papers and systematization of knowledge papers, workshops and tutorials, a poster session, and lightning talks.

Registration will open in May 2020.

Symposium Organizers

General Chair
Heather Richter Lipford, *University of North Carolina at Charlotte*

Vice General Chair
Sonia Chiasson, *Carleton University*

Technical Papers Co-Chairs
Joe Calandrino, *Federal Trade Commission*
Michelle Mazurek, *University of Maryland*

www.usenix.org/soups2020

29TH USENIX SECURITY SYMPOSIUM

Co-located with SOUPS 2020
August 12–14, 2020 • Boston, MA, USA

The 29th USENIX Security Symposium brings together researchers, practitioners, system administrators, system programmers, and others to share and explore the latest advances in the security and privacy of computer systems and networks.

The Symposium will span three days, with a technical program including refereed papers, invited talks, posters, panel discussions, and Birds-of-a-Feather sessions. Co-located workshops will precede the Symposium on August 10 and 11.

Program Co-Chairs
Srdjan Capkun, *ETH Zurich*
Franziska Roesner, *University of Washington*

Registration will open in May 2020.

www.usenix.org/sec20

Constraints and Controls

The Sociotechnical Model of Site Reliability Engineering

LAURA NOLAN

Laura Nolan's background is in site reliability engineering, software engineering, distributed systems, and computer science. She wrote the "Managing Critical State" chapter in the O'Reilly Site Reliability Engineering book and was co-chair of SREcon18 Europe/Middle East/Africa. Laura Nolan is a production engineer at Slack. laura.nolan@gmail.com

MIT's Professor Nancy Leveson gave a talk at SREcon19 EMEA about her research on safety engineering and accident analysis [1]. Leveson's work draws on case studies from military air-traffic control in Iraq, contamination of water supplies, failure to launch a satellite [2], as well as the accidents that resulted from the Therac-25 software-controlled radiation therapy device [3].

One of the examples described in that SREcon talk was a training exercise undertaken by a pair of fighter pilots. The plan was for a pilot to fire a dummy missile at another aircraft. One of the plane's missile tubes was loaded with a dummy, while other tubes contained live missiles. The pilot targeted the other aircraft, selected the tube with the dummy, and fired—a live missile. This wasn't pilot error: it was a systems accident. The plane had a smart missile selection system that would substitute another missile if the tube the pilot selected was blocked, and in this case an antenna was in front of the tube with the dummy.

The thesis of Leveson's talk is that traditional methods of managing risk in systems, such as fault tree analysis and analytic decomposition, do not work in the context of complex systems. These established techniques involve breaking larger systems down into smaller subsystems, reasoning about the likelihood of failure of these components, and calculating overall reliability of the system from there. Unfortunately, this isn't effective: many systems accidents happen because of unanticipated interactions between parts of the system that were working as intended.

We see these kinds of interactions in computer systems all the time. Reddit's outage on August 11, 2016 [4], is a great example: they were performing maintenance on their Zookeeper cluster. Reddit's autoscaler system relies on Zookeeper for input data, so in order to prevent the autoscaler from doing the wrong thing while Zookeeper was under maintenance, they turned it off. Unfortunately, their configuration management system turned the autoscaler back on, and it took their site down. That, of course, isn't as bad as shooting down a friendly aircraft, but the incidents do have elements in common.

In both those examples, no part of the system was broken, but the system overall didn't work as expected. The failure of analytic decomposition is especially acute for systems involving software, because so many software problems arise from unexpected interactions between parts of our systems, not simple component failure. Safety (or reliability, from our perspective) is a property of the entire system, not of the components of the system.

Leveson's approach, STAMP (Systems-Theoretic Accident Model and Processes), has three parts:

- Constraints, or conditions needed for the system to operate safely
- Hierarchical safety control structures, which work to enforce the constraints
- Process models describing the state of a system and how it moves from one state to another

According to STAMP, designing for reliability starts with figuring out what the key system constraints are, then analyzing how candidate designs can be controlled in such a way to be kept within those constraints. One article doesn't afford nearly enough space to do justice to the intricacies of STAMP, so this column will be focused on Leveson's concepts of system constraints and control structures and how they relate to site reliability engineering (SRE).

Hazards, Constraints, and Controls

For Leveson, safety is all about maintaining control of the system. Start by figuring out the hazards around your system. For a public water supply the hazard might be "avoid exposing the public to contaminated water." In a production software environment, the hazards are likely to be things like "keep the error rate under 0.1%," "don't expose web servers directly to the Internet," or "don't lose user data."

From the hazards, you derive a set of constraints. For the water supply system, those might be "water quality must meet standards," and "if water quality falls below standards, steps must be taken to reduce risk of exposure (e.g., boil-water advisories)."

For your production software system, constraints could be things such as "new releases must be canaried to ensure the error rate doesn't increase," "firewall rules must be in place to prevent access to the web servers," or "maintain at least three replicas of critical data," as well as "the system must have enough compute, storage, and bandwidth available to it in datacenters foo and bar," or "service foobaz, on which we depend, must be operating with a 95th percentile latency under 100 milleseconds."

This should look pretty familiar so far: these are more-or-less service level objectives (SLOs) that our system is expected to fulfill and SLOs that our system needs from other systems or infrastructure.

According to Leveson, hazards and constraints are a critically important part of system design, and deriving them needs deep domain expertise. Once you've defined your constraints, you have to figure out how to monitor them and keep your system within them. This means designing the controls that enforce the constraints. If an incident does happen, accident analysis should be focused on finding the failures or gaps in the system controls that allowed the incident to take place.

Controls are not only technical, however; the entire system of humans involved in the development, operation, and oversight of a system are also part of the control system. For some safety-critical systems, like nuclear reactors or food safety, this goes as far as including the government and courts as part of the control system. For SRE, this usually means the team responsible for a given service and the management and leadership structure to which SRE teams report.

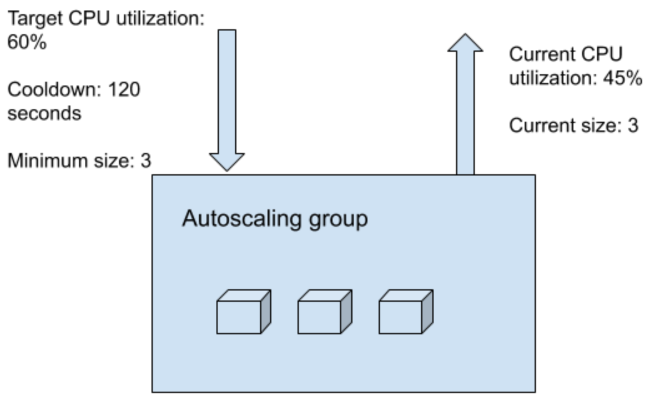


Figure 1: An autoscaling group

Reference and Measuring Channels as SLOs and SLIs

To control a system you need two things: a way to specify the constraints on the system and feedback. Take a simple technical example: an autoscaling group (as provided by the major cloud platforms).

Figure 1 is a model of an autoscaling group from the perspective of the user—an implementor would have a more detailed view of the system internals. The autoscaling group currently contains three instances. It is configured to keep a minimum of three instances running. It'll increase the number of instances if the CPU utilization exceeds 60%. There's a cool-down period of 120 seconds, so the autoscaler won't increase or decrease the number of instances until two minutes have passed since the last scaling action.

In STAMP terminology, the control information is the reference channel (the inward arrow in Fig. 1): this is the information needed to do the job of imposing constraints on the system. The outward arrow, the system metrics, is the measuring channel, which gives information about how the system is behaving—is it within its constraints or not?

The concepts of reference channels and measuring channels map very closely to SLOs and SLIs (service level indicators), respectively. An SLO, or reference channel, is a specification of how you want your system to behave, and an SLI, or measuring channel, shows whether or not your system is achieving its SLO. Control doesn't work without feedback. This, perhaps, is the reason that SLOs and SLIs are so often seen as the essential first step to adopting SRE practices—but they are definitely not the only form of reference and measuring channels needed.

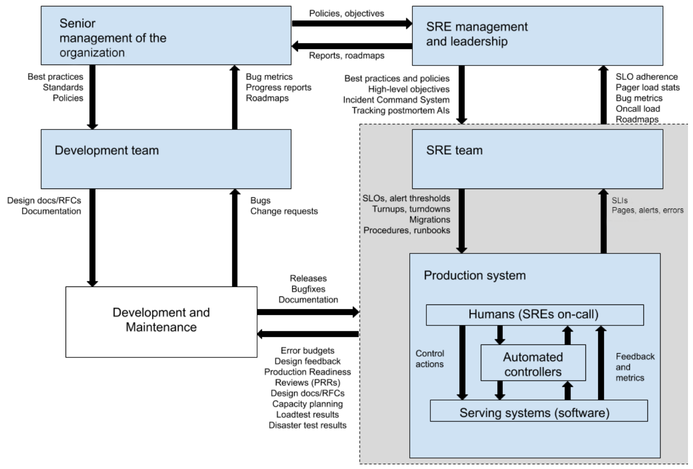


Figure 2: SRE model of sociotechnical control

The SRE Sociotechnical Model

Leveson's SREcon talk really resonated with a lot of people at the conference. The problems of complexity arising from component interactions are our everyday experience, even if our context is with RPCs or data pipelines rather than ballistic missiles or satellite launches. We're very familiar with the need for dynamically controlling the systems that we run and the difficulties that arise from that (we saw some examples in the last installment of this column when we looked at dynamic control systems and public cloud outages [5]).

The aspect of STAMP that is most relevant to SRE, however, is that it treats the organizational side of system reliability as a first-class citizen. What SREs do at a purely technical level doesn't look much different to software engineering or system administration: we do debugging and performance analysis, and we write C++ or Java or Go or bash scripts or Terraform configs or Prometheus rules, like anyone else in software. The organizational practices aimed at managing and controlling technical complexity, however, make SRE different—and it turns out that many of these practices have close analogues in STAMP.

According to Leveson, safety control structures are hierarchical. Constraints are created at a higher level to control processes at the lower levels of the hierarchy, until you eventually arrive at the operating process itself and its direct control mechanisms.

There are different ways that SRE engagements can be structured organizationally [6], but the classic setup at Google, where SRE originated, is for an SRE team to report to an SRE management function and to collaborate with one or more development teams. The SRE team manages the system in production and uses the experience gained from that to inform its engineering work, which is focused on reliability, scalability, and performance. The development team works on features and collaborates with the SRE team on changes needed to keep the system stable and within its service level objectives.

The SRE organizational model includes a host of different controls and forms of feedback, from error budgets and direct interaction with the production system itself to forms of control generally performed by management, such as setting organization-wide policies and objectives and measurements, like pager load over time. The diagram above is a SRE-specific version of Leveson's general model of sociotechnical control [2].

Constraints, Controls, and the SRE Team

At the SRE team level, the focus is on the technical systems. SRE teams are normally deeply involved in defining SLOs for their systems. Much of our technical work directly involves ensuring the system is kept within SLOs—from design work to monitoring and automation to control the system.

Healthy SRE teams also self-monitor, working at one level of abstraction above the system itself. They're looking at trends in SLIs over longer periods of time, for patterns of incidents, for upcoming problems like hitting scalability limits, for upgrades or migrations that need to be performed, for new kinds of repetitive manual work that may need to be automated.

Teams need control structures to make sure these self-monitoring activities happen regularly. Most SRE teams use a weekly production meeting [7] to review the state of their production systems, and this meeting is the natural site for much of the self-monitoring that teams do. Teams will review service metrics, outages, paging events, and other interrupts such as tickets: all of these are measuring channel activities. As a result of this, teams will make decisions that affect their reference channels: updating runbooks, tweaking alerts. They'll also surface issues that require engineering work, which might be done within the SRE team or become requests to the partner development team, which usually has some representatives in attendance at the production meeting.

SRE and Development Team Collaboration

As well as attending the weekly SRE-run production meeting, SRE teams have several other reference and measurement channels with developer teams. Development and maintenance of systems is a joint activity shared by developers and SREs. Both SREs and developers write design documents (also known as RFCs, or requests for comment) and provide feedback on the other team's designs; this is a very important pair of reference and measurement channels, as each side has its own set of system knowledge and perspectives.

Production readiness reviews (PRRs) [8] are another important channel between developers and SREs. PRRs are generally used when a new service is being onboarded by an SRE team. SRE teams normally evolve a fairly comprehensive team-specific checklist for new services that covers items such as:

- Review of system architecture and dependencies
- Review of the system against the team and the organization's standards
- Review and development of SLOs
- Review and development of monitoring and alerting
- Review of change management practices (such as canarying)
- Developing training that can be delivered to the SRE team

During the PRR process, the SRE team will work through this checklist with the developer team. The PRR process is a reference channel; the SRE team imposes constraints on the standards of the systems they are willing to support.

Error budgets are another well-known reference channel that developer teams and SREs share. Error budgets are defined based on SLOs: how much unavailability can a service have during a given quarter and still be within its SLO? The SRE team monitors a service's SLO and error budget. If the error budget for the quarter has been exhausted, then an SRE team should push back against risky launches and normally will negotiate with the developer team to prioritize reliability-related work.

Monitoring SRE Teams

Leveson says control is hierarchical. We've already seen how SRE teams control and monitor their services. In large organizations, SRE management and leadership should also have a role to play in monitoring the health and efficiency of SRE teams:

- Are their services generally meeting their SLOs?
- Are they getting paged too often?
- Do teams have sufficient staffing to do substantial engineering work as well as operational work?
- Are high priority postmortem action items being done?

This doesn't mean that leadership should micromanage. The feedback loops provided by measurement channels get longer the further up any hierarchy you go, and so control becomes less effective. Management should be concerned with longer-term patterns over multiple quarters.

This should not be a coercive approach, focused on demanding that teams hit their metrics by working unsustainable hours or at the cost of doing the right thing for their service—for instance, teams should be able to prioritize fixing a newly found major risk to their service's stability over low and medium-priority postmortem action items, even if it means that those open postmortem action items will be visible to management in the form of metrics. The approach should be about making sure that teams have resources and organizational support to get their job done effectively and to prioritize the highest impact work. Done right, this should not be a box ticking exercise.

SRE management is also in a great position to increase the effectiveness of the entire SRE organization by spotting places where standard tools and processes can help—these are, of course, reference channels. Examples of this could be introducing a standard process for managing incidents, or kicking off a project to build a production-grade tool for doing deployments or chaos engineering.

Conclusion

This article has just scratched the surface of Leveson's work. Nevertheless the STAMP concepts of reference and measurement channels and hierarchical control systems very closely describe what it is that SREs do. Learning about STAMP gave me a clearer insight into the organizational side of SRE.

The "what is the difference between SRE versus DevOps" debate has been well played out by now, but I'll add my contribution nonetheless: SRE is about the humans that design and control the systems as much as it is about technical considerations.

SREs are in the business of defining objectively which system states are acceptable and which are not. Our job is implementing controls, both technical and organizational, to keep our systems healthy. Our teams are part of those systems too, and also need to be healthy to be effective. Pain is unpleasant, but it is an essential form of feedback—it tells us to stop doing the thing that hurts in order to stay healthy. Far too many teams in operations are in pain, quarter to quarter, year to year. Does your organizational model notice?

References
[1] N. G. Leveson, "A Systems Approach to Safety and Cybersecurity," SREcon19 EMEA: https://www.usenix.org/conference/srecon19emea/presentation/leveson.

[2] N. G. Leveson, *Engineering a Safer World: Systems Thinking Applied to Safety* (MIT Press, 2012).

[3] N. G. Leveson, "Medical Devices: The Therac-25": http://sunnyday.mit.edu/papers/therac.pdf.

[4] "Why Reddit Was Down on Aug 11": https://www.reddit.com/r/announcements/comments/4y0m56/why_reddit_was_down_on_aug_11/.

[5] L. Nolan, "Managing Systems in an Age of Dynamic Complexity Or: Why Does My Single 2U Server Have Better Uptime than GCP?" *;login:*, vol. 44, no. 4 (Winter 2019): https://www.usenix.org/publications/login/winter2019/nolan.

[6] D. Ferguson and P. Labhane, "SRE Team Lifecycles," in B. Beyer, N. R. Murphy, D. K. Rensin, K. Kawahara, S. Thorne, eds., *The Site Reliability Workbook: Practical Ways to Implement SRE* (O'Reilly, 2018).

[7] N. Murphy et al., "Communication and Collaboration in SRE," in B. Beyer, N. R. Murphy, D. K. Rensin, K. Kawahara, S. Thorne, eds., *Site Reliability Engineering: How Google Runs Production Systems* (O'Reilly, 2016).

[8] A. Cruz and A. Bambhani, "The Evolving SRE Engagement Model," in B. Beyer, N. R. Murphy, D. K. Rensin, K. Kawahara, S. Thorne, eds., *Site Reliability Engineering: How Google Runs Production Systems* (O'Reilly, 2016).

Python and Memory

PETER NORTON

Peter works on automating cloud environments. He loves using Python to solve problems. He has contributed to books on Linux and Python, helped with the New York Linux Users Group, and helped to organize past DevOpsDays NYC events. In addition to Python, Peter is slowly improving his knowledge of Rust, Clojure, and maybe other fun things. Even though he is a native New Yorker, he is currently living in and working from home in the northeast of Brazil. pcnorton@rbox.co.

I found myself thinking about an interesting problem I ran into a few years ago. I was wondering why an open source metrics collection system seemed to have a relatively low performance ceiling when relaying metrics.

After much troubleshooting, I found that the performance issue resided in attempts at splitting a list: when it had thousands of messages, it would pull off some messages from the front of a list, then split the list, and the way it was doing this was inefficient. In the end I found some improvement in using a deque, but the problem has left me with continuing questions about some of these oddities in how Python does its memory management.

Since then I have found myself with an imperfect and incomplete understanding of Python's memory management, and I thought it would be interesting to take a quick look at the standard library to see if it can help tell us what Python is doing—how we allocate and manage memory. Let's start with a very light-on-details version of how we got here.

Back in the Day...

In the days before Python 2.0, Python's memory management simply consisted of a method called *reference counting*, that is, most objects referenced from somewhere—within the global scope in a file, function, or object or class from a module you've loaded—will maintain a field that the interpreter will increase when there is a new reference to the object and will decrease when a reference is removed (e.g., a context manager going out of scope or a function finishing up). For many data types, that's pretty foolproof, and it's a very simple system for a language runtime to implement.

Circular Data Structures

The well-known weakness in this simple solution is that there are lots of situations where, deliberately or incidentally, we can create circular references—object A contains a reference to object B, and object B also contains a reference to object A. In case you're having trouble visualizing the situation, something like the following serves as a trivial example:

```
dict_A = {
    "a": "this is an A",
    "b": dict_B
}
dict_B = {
    "a": dict_A,
    "b": "this is a B"
}
```

When we create situations like this, it's called a *reference loop*, and since both objects will never have a reference count that goes to zero upon going out of scope, it is now memory that can't be collected automatically by a reference-counting garbage collector (GC).

As you can see in this example, you need a so-called container type to do this. Lists and dictionaries are primary examples of container types, which are so-called because they contain references to other objects. Because they're more complex than simple non-container types like a bool, integer, float, or string, and because they are essentially a way to create memory pointers, by their nature they can break reference counting if not used with extreme care.

Python Needed More than Reference Counting

Python made it to version 2 with the reference counter as its only garbage collection mechanism, and by then its limits were well known. Starting in version 2.0, an additional garbage collection mechanism, called a *generational collector*, was added that could clean up where the reference counting couldn't. The principle is that it looks for objects that are allocated but possibly not reachable by the reference counter and makes sure that any objects that aren't in use are cleaned up. When they are found to be in use, it makes the sensible decision that something that's in use is likely to remain in use, and "promotes" that object to a more senior generation, which doesn't get checked as often.

Reference counting is very fast and easy to implement, while the generational collector is more complex: because it scans memory and needs the state of memory to not change while it's running, it has more of an impact on performance because every time it runs it must stop the main thread, grab the Global Interpreter Lock, and do its work.

The more objects that the generational garbage collection needs to manage, the longer it will take to scan memory. The idea is that most objects will be immediately handled by the reference counter, making the more expensive alternative the one that will be used less.

How Active Is the GC, Really?

You can get a precise idea of how much work the GC is doing by using the `gc` module.

From version 3.3 on, the Python's `gc` module has provided hooks that will invoke a callback to notify the program whenever a GC event occurs. With these hooks, you can gather the data you need to describe what the collector is doing and to help you infer why it's doing it. Most of the work that the GC is doing is usually invisible, but this module helps with the important feature of letting you see how fast the interpreter thinks it needs to clean up after itself, and how actively it is doing so.

```
import gc
print(gc.get_stats())
def print_hook(phase, info):
    print(f"The gc hook is in phase {phase}")
    print(f"And the gc hook provided this info: {info}")

gc.callbacks.append(print_hook)
foo = list(range(500))
del(foo)
print(gc.get_stats())
```

And at the end you should see a summary provided by `gc.get_stats` that roughly matches what you see as having been collected and shown by the `print_hook`.

Once you are able to know more about the garbage collection patterns that code is creating, you may find yourself wondering what kind of controls you have over how the garbage collection actually works—for example, can you schedule the garbage collection for times that you prefer? Or are there other ways to control the garbage collection behavior at all?

There are tunable knobs that allow you to manage your programs' garbage collection. For instance, you have the `gc.disable` and `gc.enable` to guarantee that your code runs uninterrupted for a span. Or if you know for sure that you won't need it, you can just disable automatic garbage collection at the start—for example, if you have a job that loads a lot of data, then outputs some data, and exits after a short time, who bothers with garbage collection at all? In some cases it can be a huge advantage to not clean up garbage before exiting.

Similarly, if you find you need to manually manage when collections are run, you can in fact make the garbage collector run a collection using `gc.collect()`, which will run either a full collection (check everything) or you can tell it to work on a particular generation by specifying 0, 1, or 2.

If it turns out that there are sections of your code that legitimately will never change, or if a lot of work is done, then a fork-exec is done: the Python `gc` module provides the `gc.freeze()` and `gc.unfreeze()` functions, whose main use is documented as being for forking a process and minimizing churn in the VM subsystem having to map child processes to new pages if the parent process decides to collect something that it didn't need to (after all, often it will be doing nothing but using `wait()` in a loop).

Thresholds

Thresholds are another interesting setting. If you think you want to collect more or less frequently but still have collection be automatic, you set thresholds, which represent how many times container objects are allocated until the GC kicks in and does a run over first or second generation objects to see what needs collection. The thresholds set how many objects there are in a particular generation before a garbage collection run is initiated. The default thresholds of 700, 10, 10 show the expectation that there will normally be a lot more small objects created that will be collected than there are that will survive, and that only a few will be tenured and survive to be shifted from the 0th generation to the 1st and 2nd. The 2nd generation is the highest and shouldn't have very many unreachable but live objects.

The idea with thresholds is that if you discover that you've created a lot of garbage that is getting tenured, and should remain live, you can adjust those thresholds to prevent the garbage collector from doing unnecessary work.

Tracemalloc—Seeing Which of Your Files Are Allocating Memory

On the other side of the coin, what's also easier to observe in Python 3.3+ is what's happening on the allocation side. It's conceptually clear that whenever you create an object, there will be memory allocated, but the layers of modules, objects, iterators, etc. can often make it hard to just take a glance at your code and have a good idea as to how much work is being put into creation and allocation of memory.

To do that there is an interesting module called `tracemalloc` [1], which can be used to show you where your allocations are happening. And while tracing, you can even pick up individual objects and see where their allocations took place, so you can figure out where in your code you may be exercising the allocator. If you're doing enough small allocations, this module can help you confirm whether memory is being used where you expect it. This seems like it would be most useful when understanding code from outside modules but could still be useful in code you've written for yourself.

The sample here is a variation on the standard library documentation and will show you how many allocations were made in each file that was recorded (though the limit of `[0:10]` when extracting from the traceback will only show us the top 10 in this case):

```
import tracemalloc
import requests

tracemalloc.start(25)
resp = requests.get('https://google.com/')
t = tracemalloc.take_snapshot()
tracemalloc.stop()
print("\n".join([str(s) for s in t.statistics('traceback')
[0:10]]))
```

This second example will do something very similar but shows you what the stack looked like when a particular object was allocated its memory:

```
import tracemalloc
import requests

tracemalloc.start(25)
resp = requests.get('https://google.com/')
objinfo = tracemalloc.get_object_traceback(resp)
tracemalloc.stop()
print("\n".join([str(l) for l in objinfo]))
```

In this case, the output should look something like this:

```
$ /usr/bin/Python3 gc-test-obj-traceback.py
gc-test-obj-traceback.py:5
/usr/lib/Python3/dist-packages/requests/api.py:75
/usr/lib/Python3/dist-packages/requests/api.py:60
/usr/lib/Python3/dist-packages/requests/sessions.py:533
/usr/lib/Python3/dist-packages/requests/sessions.py:668
/usr/lib/Python3/dist-packages/requests/sessions.py:668
/usr/lib/Python3/dist-packages/requests/sessions.py:247
/usr/lib/Python3/dist-packages/requests/sessions.py:646
/usr/lib/Python3/dist-packages/requests/adapters.py:533
/usr/lib/Python3/dist-packages/requests/adapters.py:265
```

And sure enough, in my Python installation adapters.py on line 265 is the beginning of the `build_response()` function in the requests module, and that is where the object was created. I was very pleasantly surprised to find this particular behavior. It seems to be a great tool to help with code spelunking to discover where an object actually came from, which can be very dependent on runtime conditions when you're creating objects in complicated situations.

The existence of the `tracemalloc` module is interesting, and it would be fun to explore more of its API and expected behaviors by pointing it at live code.

The More State-of-the-Art Allocators

However fun it is to be able to look under the hood, it wouldn't be fair to not mention a bit more about what the current state of the art is outside of Python.

Since the addition of the generational collector, a lot of research has been done in the garbage collection field. Most of the practical research that I'm familiar with has focused on the use case of Java and the JVM. So I'll give an overview of my incomplete understanding of the progress of Java's last few generations of garbage collection from the point of view of someone who's had to struggle with it.

Most of us have probably used the Concurrent Mark+Sweep collector, which was the primary Java garbage collector for a long time. However, it didn't age well—as applications and platforms switched from 32-bit pointers to 64-bit pointers, developers of applications that were memory intensive found it was fairly common to experience multisecond stop-the-world GC pauses at the most inconvenient times. In addition, the more memory that was in use, the worse the impact of the pauses and the longer the pauses. And, as the available memory in 64-bit systems has grown, the CMS collector has shown its age and was not able to keep up.

In the past decade a new garbage collector known as the G1 GC matured in the Sun/Oracle Java 8 with the goal of reducing pause times and enabling the JVM to be able to handle larger memory heaps, and it no longer became a very tricky proposition to request and use a heap of greater than eight GB.

Currently, JVM-hosted applications are growing to use more resources and more memory in specific. In the OpenJDK era there are two brand new garbage collection systems that have miraculously grown to handle more memory with smaller, less-noticeable pauses. One is called Shenandoah [2] and the other is called `zgc` [3]. Both are incredibly ambitious and featureful. Users of Java will be very happy with this change.

In short, the state-of-the-art of memory management and garbage collection has continued to advance. From my perspective, the driver in garbage collection research has been Java, which has made steady gains. Python is already used in projects that have large memory footprints, but anecdotally I haven't heard that it's great, and I wonder whether advances in the language, like the multiple interpreters in PEP 554 [4] planned for Python 3.9, are likely to have an impact in memory usage by making high-performance multithreaded Python applications start to seem more tractable.

In any case, please go and try out these modules on your own code, and enjoy!

References

[1] https://docs.Python.org/3/library/tracemalloc.html.

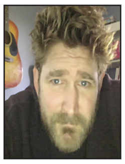
[2] https://wiki.openjdk.java.net/display/shenandoah/Main.

[3] https://wiki.openjdk.java.net/display/zgc/Main.

[4] https://www.Python.org/dev/peps/pep-0554/.

iVoyeur
eBPF Tools: What's in a Name?

DAVE JOSEPHSEN


Dave Josephsen is a book author, code developer, and monitoring expert who works for Fastly. His continuing mission: to help engineers worldwide close the feedback loop.
dave-usenix@skeptech.org

There is a story in ancient Egyptian folklore that the goddess Isis created a serpent to poison the sun god Ra. Isis withheld the antidote from the withering sun god in exchange for his true name, which he eventually surrendered. This—the true name of Ra—gave Isis complete power over him and enabled her to elevate her son Horus to the Egyptian throne.

Horus died with Ancient Egypt, but Isis lived on into Greek mythology, along with many of her Egyptian counterparts. In fact the Romans were still building temples to her ~1000 years later. I find this early story about her gleaning Ra's true name kind of fascinating because she also happens to be one of the very few gods who was never renamed in the whole of human history. Ra, of course, became Apollo, who in turn became Phoebus Apollo to the Romans.

So all the while the Egyptian gods were being given Greek names, the Sumerian gods were being given Akkadian equivalents, and throughout the infamous Roman divinity-rebranding pivot from Greek mythos, Isis remained Isis. It's almost as if her nearly prehistoric cognizance of the power inherent in names somehow rendered her immune from the incessant attempts of mortals to relabel the divine.

Today, our god situation is comparatively simple (in cardinality at least), but our complicated relationship with names lives on. There is, for example, a Sunni Hadith (https://sunnah.com/bukhari/80/105) that asserts God has 99 names, and to know them is the path to paradise. The power of the "true name of God" is a central theme in Kabbalism, Sufism, Judaism, and in Christianity where we're reminded not to use it in vain, and where we find Jacob wrestling with an angel who refuses to reveal his true name.

Richard Feynman famously doubted the significance of names when he wrote about the difference between naming a thing and knowing it (https://fs.blog/2015/01/richard-feynman-knowing-something/). "See that bird?" he said. "It's a brown-throated thrush, but in Germany it's called a halzenfugel, and in Chinese they call it a chung ling and even if you know all those names for it, you still know nothing about the bird. You only know something about people; what they call the bird."

If naming something corporeal like a bird provides us no useful insight, what then are we to make of our propensity for foisting names upon the divine and ethereal? This is a question Socrates ponders in Cratylus; are names arbitrary labels? Or might they carry within them some innate, visceral power beyond our ability to comprehend? Are names the random vocalizations of apes or priceless gifts from some immortal creator?

There's a joke in our industry that goes: "There are two hard problems in computer science: cache invalidation, naming things, and off-by-one errors," and I personally would reorder that list such that naming things came first. There is, you know, a positively terrifying undercurrent to the act of giving something a name. A nagging suspicion that what I'm doing is not naming a thing at all but, rather, foisting upon future generations of engineers the banal and loathsome historical context of the present.

```
Device:  rrqm/s    wrqm/s     r/s        w/s      rkB/s       wkB/s  avgrq-sz  avgqu-sz  await   r_await  w_await  svctm  %util
sda        0.00     10.00    0.00      22.00       0.00      134.40     12.22      0.00   0.00      0.00     0.00   0.00   0.00
sdb        0.00      0.00    0.00       0.00       0.00        0.00      0.00      0.00   0.00      0.00     0.00   0.00   0.00
sdd        0.00   4275.40   13.60    7349.00      54.40    47689.60     12.97     21.10   2.87      0.53     2.87   0.08  57.76
md0        0.00      0.00  279.60   63568.20    1118.40   259052.00      8.15      0.00   0.00      0.00     0.00   0.00   0.00
sdc        0.00   4266.40   22.80    7343.80      91.20    47625.60     12.95     27.58   3.70      0.07     3.71   0.08  60.88
sde        0.00   4190.40   36.20    5611.60     144.80    39660.80     14.10      4.78   0.85      0.15     0.85   0.07  38.72
sdf        0.00   4189.20   20.80    5612.80      83.20    39660.80     14.11      4.34   0.77      0.23     0.77   0.06  34.56
sdo        0.00   4261.60   27.00    7508.40     108.00    48224.00     12.83     28.31   3.76      0.33     3.77   0.08  58.64
```

Extended disk report, trimmed to a reasonable length

Consider sed, a shell tool that derives its name from a still-older tool, ed (https://en.wikipedia.org/wiki/Ed_(text_editor)), developed in August 1969 when memory was so dear a commodity that every computer program had two- and three-letter names. Or Kubernetes, a tool with so unwieldy a name that the community has resorted to numeronyms (https://en.wikipedia.org/wiki/Numeronym) to deal with it on a daily basis.

eBPF

Despite the considerable buzz surrounding eBPF these days, it's completely understandable if you're not exactly sure at first blush just what the heck it actually *is*. For one, it carries an understated—some would even say misleading—name, which like many things named by engineers, has more to say about its origins than its identity. I say it "carries" its name, but really it drags its name behind it like an iron ship-anchor. A name that makes it impossible to introduce to newcomers without delving into the history of its origins.

Upon hearing that the acronym eBPF stands for "Extended Berkeley Packet Filter," you might come to the conclusion that it's a packet-filtering program, which is either mostly wrong or completely wrong, depending on what you expect to get out of a name. If you think a name should imply what a thing is, you're completely wrong. eBPF is not a packet-filtering program; it's a register-based Virtual Machine running inside the Linux Kernel.

If you think a name should imply what a thing is *good for*, then you're only mostly wrong. eBPF can, in fact, filter packets for you. But it can also do many, many other things for you that have nothing whatsoever to do with the network stack.

Just the other day, in fact, I used an eBPF program to identify a failing drive in an mdraid array by asking it for a histogram of block-I/O latency as a function of device. One drive in the array had actually already failed and had been replaced with a new drive. But having added the new drive and rebuilt the array, disk I/O was still noticeably slow.

This left me in the unenviable position of having an array of 12 disks, one (or some) of which were not performing as well as they should. I don't know about you, but when I've encountered problems like this in the past, I've turned to iostat.

iostat -dx5

Sometimes I wonder how many hours of my life I've spent staring at the output from this little command, which shows an extended disk report similar to the one above every five seconds.

The input data for this report comes from /proc/diskstats and is documented in the kernel docs https://www.kernel.org/doc/Documentation/iostats.txt. If you skim it, you'll probably notice that the report format and other details depend on the kernel version you're running, which is annoying. If you put your engineer hat on and read in a bit deeper, you'll start to come across some weird details related to—of course—*naming*.

The avgqu-sz field, for example, is misleading in that it isn't really an average of the queue size, because it doesn't show how many operations are queued waiting for service. Rather it shows how many I/O ops were either in the queue waiting *or being serviced*. Similarly await is not an in-queue wait time but actually measures end-to-end latency. Oh, and the disk report's last column %util? It tells you how much of the time during the measurement interval the device was in use (many people would understandably interpret something called "%util" as a measure of whether a device is reaching its limit of throughput, but nope).

If you know these things (and more) about iostat, and you are practiced at staring at this output, and you have something of a baseline understanding of what a healthy I/O load looks like for your system, and you have fewer than 50 disks, iostat will probably get you where you need to be. It probably would have gotten me to the finish line with my latency problem eventually, but I'd been reading about eBPF on Brendan's blog (http://www.brendangregg.com/blog/2019-01-01/learn-ebpf-tracing.html), and I found myself staring at iostat and wondering whether there was a BPF tools script that could show me a breakdown of how much latency each individual disk was experiencing. Check this out:

```
usecs               : count    distribution
    0 -> 1          : 0        |                                        |
    2 -> 3          : 0        |                                        |
    4 -> 7          : 0        |                                        |
    8 -> 15         : 0        |                                        |
   16 -> 31         : 6870     |                                        |
   32 -> 63         : 516091   |****************                        |
   64 -> 127        : 838139   |*****************************           |
  128 -> 255        : 963522   |*********************************       |
  256 -> 511        : 318996   |*************                           |
  512 -> 1023       : 146827   |******                                  |
 1024 -> 2047       : 74222    |***                                     |
 2048 -> 4095       : 66658    |**                                      |
 4096 -> 8191       : 33339    |*                                       |
 8192 -> 16383      : 25817    |*                                       |
16384 -> 32767      : 13587    |                                        |
32768 -> 65535      : 8990     |                                        |
65536 -> 131071     : 425      |                                        |
```

This output, a histogram of I/O latency, came from the `biolatency` tool in the BCC tools suite (https://github.com/iovisor/bcc). Biolatency, read: block I/O latency, even has a name I can get behind. Passing a "-D" gleans a histogram breakdown of latency per disk. Where does this awesome biolatency tool get its data, you ask? Well from the enhanced Berkeley Packet Filter obviously!

Wait, what?

eBPF works like an embedded `lua` interpreter or the spidermonkey VM that executes JavaScript inside the Mozilla web browser. It resides in kernel space, ready to execute bytecode supplied from userspace. Its original intent was to filter packets without having to resort to context-switches, but it has grown to become a fully fledged kernel tracing system comparable to DTrace in long-lost Solaris.

A userspace BPF script sends a bytecode to the kernel together with a program type which determines what kernel areas the program can access. If you look at the source code for `biolatency` (https://github.com/iovisor/bcc/blob/master/tools/biolatency.py), you'll notice it is a Python program which contains a small C program inside it as a string (starting on line 56).

The Python code takes care of compiling and loading that block of C code into the kernel and then stays resident in memory, collecting data from its own kernel probe, and eventually presenting it to us, the user. I'm intentionally glossing over *a lot* of detail here, including a pre-load code verifier which guarantees your probe payload won't crash the system. There are a lot of moving parts, but the result is high-resolution, low-cost visibility into the inner-workings of the system and everything running on it. Unprecedented observability.

I'd like to spend the next few articles together digging into eBPF more deeply. My plan is to use our new friend, the biolatency tool, as a laboratory frog we can dissect together. We'll start light, talking about the various endpoints eBPF gives us to get our hooks into the kernel, and finish up with hopefully a solid place to get started crafting your own eBPF programs. Who knows, maybe we'll even filter some packets.

Take it easy.

Simplifying Repetitive Command Line Flags with viper

CHRIS "MAC" MCENIRY

Chris "Mac" McEniry is a practicing sysadmin responsible for running a large e-commerce and gaming service. He's been working and developing in an operational capacity for 15 years. In his free time, he builds tools and thinks about efficiency. cmceniry@mit.edu

In "Knowing Is Half the Battle: The Cobra Command Line Library of Go" [1], we explored using the `github.com/spf13/cobra` library for creating command line tools. In this article, we're going to expand on that by hooking in multiple ways to handle the flags to those commands by using a sister library to `cobra: github.com/spf13/viper`.

How we handle command configuration changes over the lifetime of the tool. A common evolution for handling command configuration is, in order of precedence:

- command line supplied flag
- environment variable
- configuration file

When you first start to use a tool, you will typically supply the flags on the command line. This allows you to explore and iterate with the flags easily.

After you get comfortable with them, you'll want to avoid having to reenter any common values. For example, `--user` or `--server` become very repetitive if you have to enter them every time you run the command. This is the perfect place for environment variables to come into the picture. Set the environment for your shell session, and you can skip setting it on the command line each time.

Eventually, you're comfortable enough with the overall setup to commit those configurations to a file to preserve them over multiple sessions. These typically end up as part of your dotfiles. You set the file and never have to configure your environment or command line again.

Yes, sometimes you skip steps so this pattern is not exclusive, but it is especially common in tool development.

Since the tool configuration is built up this way, all three layers of configuration methods are available throughout. There are two additional benefits that fall out of these configuration methods:

- You can temporarily override the values from the environment or command line. This allows you to test out new configurations without changing your defaults.
- Different runtime environments and setups prefer different formats. For example, your Puppet setup may prefer configuration files, your Dockerfiles setup may prefer environment variables, and your Kubernetes setup may prefer command line arguments. A flexible binary supports multiple environments since it can support all three mechanisms. This last part is especially apt for 12-factor applications.

We're specifically using the `viper` library because it builds upon the work of the `cobra` library from the previous article. This combination follows the precedence order identified above. This only holds for flags (`--flag`) and not for full command arguments. Arguments are typically specific to each command invocation, and it is unusual to encode this in environment variables or configuration files.

The code for these examples can be found at https://github.com/cmceniry/login in the "viper" directory. Each directory corresponds to a section below and should be executed using `go run $DIR/main.go` to follow along with the article. This uses `go` module support (minimum Go version 1.11), so no prep work is required once the repository is cloned.

Default

To establish a baseline, we're going to set a default for `viper`-maintained values (this will also help to build up the scaffold around the examples). As usual, we begin with the standard Go intro—setting up our main and imports.

default/main.go: intro.

```
package main

import (
    "fmt"
    "github.com/spf13/cobra"
    "github.com/spf13/viper"
)

func main() {
```

We're going to be building on top of the existing `cobra` command. In our `Run`, we're going to just print the output of our flag. Specifically, we get the configuration value of the `Flag` item and we will get it as a string (or nothing).

default/main.go: cobra.

```
rootCmd := &cobra.Command{
    Run: func(c *cobra.Command, args []string) {
        fmt.Println(viper.GetString("Flag"))
    },
}
```

Setting a default in `viper` is a single function `viper.SetDefault`.

default/main.go: viper.

```
viper.SetDefault("Flag", "default")
```

And to round it out, we execute into our `cobra` command.

default/main.go: execute.

```
rootCmd.Execute()
```

With all of that together, we can run our tool and get our internally set value for `Flag`.

```
$ go run default/main.go
default
```

Command Line

Now let's add the first pattern by pulling the value in from the command line flag. The code here will be identical to the `default` case, but we're going to add a couple of lines just before the `Execute`. These set up the command line flag (which comes from the `cobra` command as in the *;login:* article [1]) and then bind it to the `viper` configuration.

commandline/main.go: flag.

```
rootCmd.Flags().String("flag", "", "help for flag")
viper.BindPFlag("Flag", rootCmd.Flags().Lookup("flag"))
```

We can demonstrate that by just adding these lines, we maintain our default compatibility, but we also add support for our command line flag.

```
$ go run commandline/main.go
default
$ go run commandline/main.go  --flag cli
cli
```

Environment Variable

The next step in our flag handling evolution is to set this using an environment variable. As previously, this is done with the addition of a few more items before our `cobra` execute. The first function creates a pseudo-environment namespace so that we don't accidentally conflict with other applications. The second function connects the environment variables with the `viper` configuration. Make special note that `viper` connects them with the convention of all uppercase with prefix, so in this case, `VF_FLAG`.

```
viper.SetEnvPrefix("VF")
viper.BindEnv("Flag")
```

With these in place, we can now use the default, environment, or command line.

```
$ go run envvar/main.go
default
$ VF_FLAG=env go run envvar/main.go
env
$ VF_FLAG=env go run envvar/main.go  --flag cli
cli
```

Configuration File

`viper` supports a variety of configuration file formats and even has autodetection for them. For simplicity, we're going to go with the TOML format:

```
Flag = "configfile"
```

As before, we're building on top of the previous examples by adding a few lines before executing our `cobra` command. First, we tell `viper` where to look for the configuration file. Next, we tell it which configuration file to use (notice that the suffix is ignored since we're using autodetection). And, finally, we read the config file. This is the first call that can produce an error. To support compatibility with the other three examples, we ignore it if the file is not found and panic otherwise.

configfile/main.go: configfile.

```
    viper.AddConfigPath(".")
    viper.SetConfigName("config")
    if err := viper.ReadInConfig(); err != nil {
        // Only error on errors other than file not found
        if _, ok := err.(viper.ConfigFileNotFoundError);
!ok {
            panic(err)
        }
    }
```

Now we run it again and get our expected output. As before, we can test it with the environment and command line flag options and also still receive the expected outputs. However, short of removing the config file, we will not be able to see the default value (but you can remove the file and try as you want).

```
    $ go run configfile/main.go
    configfile
    $ VF_FLAG=env go run configfile/main.go
    env
    $ VF_FLAG=env go run configfile/main.go  --flag cli
    cli
```

Combining Multiple Configurations

In this, we used `viper` as a monolithic config. There are times when you want to break this out, and that means creating a `viper.Viper` struct (using `New`) instead of the default struct invoked by the package static funcs as we've done here. This allows you to even use it in libraries to combine configuration functionality without having to support multiple formats. To avoid conflicts, you'll want to apply judicious use of the `SetEnvPrefix` and `SetConfigPath` or `SetConfigName` functions for each configuration.

Conclusion

With just a few lines of setup, the `viper` library has given us fast configuration handling. This supports the regular model of command line flags, environment variables, and configuration files.

I hope this article has provided you with a concrete handle to the `viper` library and that this helps you in your tool development. Happy Going!

Reference

[1] *;login:* vol. 43, no. 2: https://www.usenix.org/system/files/login/articles/login_summer18_09_mceniry.pdf.

For Good Measure

Cyberjobsecurity

DAN GEER

Dan Geer is the CISO for In-Q-Tel and a security researcher with a quantitative bent. He has a long history with the USENIX Association, including officer positions, program committees, etc. dan@geer.org

Five years ago, I focused this column on jobs in cybersecurity and how they compared to the market at large. This column is a revisit with some comparisons.

The cost of anything is the foregone alternative. Cybersecurity is fraught with foregone alternatives—what do I/you get done paired with what I/you pushed aside so as to get at least something done. Five years ago, I wrote that "automation is moving beyond the routinizable to the non-routine by way of the tsunami of ever bigger data." It hardly needs saying that the above is even more true now both in terms of coverage (areas of application) and velocity of change. Machines that are cheaper than you, that make fewer mistakes than you, that can accept any drudgery that risk avoidance imposes, etc. are coming on.

What does that have to do with cybersecurity? Cybersecurity is perhaps the most challenging intellectual profession on the planet both because of the rate of change and because your failure is the intentional work product of sentient opponents. Can automation help with that? Of course and it already is, as you well know regardless of your misgivings about whether anomaly detection will work in an ever more "personalized" Internet where one man's personalization is another man's targeting. So where do "we" fit in the jobs picture?

For comparability, I am going to stick with the same data sources as last time, largely the US Bureau of Labor Statistics. BLS annually predicts [1] the 20 occupations with the best outlook for new jobs over the next 10 years, which includes both the number of jobs to be added over the coming decade and the median pay at the time the prediction is made. Multiplying the predicted number of new jobs by the then current median pay might be said to give a societal investment or cost figure for that particular job.

Figure 1 recaps BLS's values for six years ago (six because of publication schedules) when BLS predicted the decadal job gain for those top 20 occupations to be 5.9 million jobs with a median pay of $32,468 for a decadal cost of $95 million for those top 20 occupations.

For each of the 20 jobs, Figure 1 plots that job's percentage of the 5.9 million new jobs against that job's percentage of the $95 million decadal cost in the aggregate. The three more extreme are labeled: 580,800 personal care aides (9.9% of new jobs) earning $19,910 (thereby contributing 6.1% of the aggregate decadal cost), 526,800 (9.0%) registered nurses earning $65,470 (18.1%), and 244,100 (4.2%) general managers earning $95,440 (12.2%).

Figure 2 is the same scheme and scale, but now using the most current (2018) BLS data.

For 2018, some outliers are the same and some are different: 881,000 personal care aides (19.2%) earning $24,020 (11.1%), 640,100 food prep/servers (14.0%) earning $21,250 (7.1%), 371,500 registered nurses (8.1%) earning $71,730 (13.9%), 241,500 software developers (5.3%) earning $103,620 (13.1%), and 165,000 general managers (3.6%) earning $100,930 (8.7%).

Six years ago, software developers didn't even make the list so there is nothing to compare to. Looking at the other four extrema [2], BLS predicts personal care aides to have a +8.4% CAGR (compound annual growth rate) in new jobs per year over the next decade, and there has been an inflation-corrected +2.8% CAGR in their pay over the last six years. For food

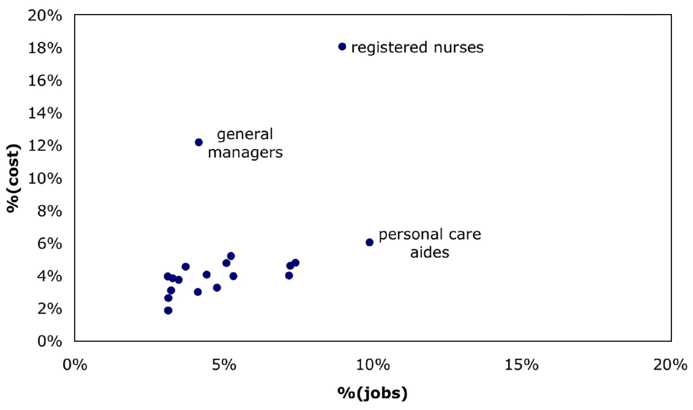


Figure 1: Percent of new labor cost versus percent of new jobs, 2012–2022

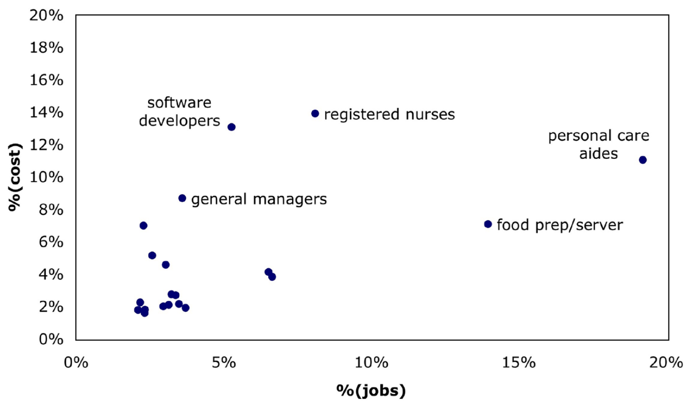


Figure 2: Percent of new labor cost versus percent of new jobs, 2018–2028

prep/servers, there will be a +3.2% CAGR in total new jobs per year, and there has been +2.2% CAGR in their pay. For registered nurses, there will be a +1.8% CAGR in jobs, and there has been a +1.2% CAGR in their pay over the last six years. For home health aides, there will be a +2.3% CAGR, and there has been a +2.2% CAGR in their pay over the last six years. Overall, the CAGR for the predicted rate of new job creation by and amongst the top 20 overall is –2.5%, that is, the top 20 will not collectively grow as quickly as they have been. Nevertheless, the CAGR for pay for those jobs overall has been +4.2%. This perhaps points to concentration of job and wage growth spreading out from the top 20 to elsewhere in the economy. Choosing what to do with your life is not getting simpler.

On the world scale, these top 20 are good jobs. The $21,250 for food prep/servers puts them in the world top 10%, the $24,020 for personal care aides puts them in the world top 8.6%, the $71,730 for registered nurses puts them in the world top 0.9%, and the $100,930 for general managers puts them in the world top 0.3% as does the $103,620 for software developers [3]. In any case, that's the spectrum of the whole US economy.

High paying jobs are precisely the ones that automation most wants to take. Turning to more interesting BLS data, namely that for "Information Security Analysts" (ISAs) [4], BLS says that today (2018) there are 112,300 of us/them with median income of $98,350 per year, putting ISAs in the top 0.4% on the world scale. Six years ago, the figures were 75,000 ISAs with mean income of $86,070 per year, so that's a +6.7% CAGR for the total number of ISAs and a +2.2% CAGR for their pay. Looking ahead, BLS predicts an additional 35,500 ISAs by 2028—a more modest job growth CAGR of +2.8% which rate of increase nevertheless qualifies ISA as the sixth fastest growing of all US occupations (after solar photovoltaic installers, wind turbine service technicians, home health aides, personal care aides, and occupational therapy assistants). For comparison, six years ago the ISA occupation was growing 16th fastest and now it is 6th fastest.

Of those 20 fastest growing jobs, only physician assistants, nurse practitioners, mathematicians, and software developers make more pay than ISAs (from +4 to +10% more). Computerworld's survey [5] confirms the pinnacle status of information security practitioners, putting a CSO (at $173,300) in the world top 0.1% (up from the top 0.2% six years ago).

So, is automation gunning for the ISA role? If not, is it because ISAs are too few to bother with, not enough people are willing to be one, or is it that the job is too hard to automate (yet)? Universities and the White House like to say that as machines take over existing jobs, new opportunities are created for those who "invest in themselves." This has been argued over and over; there isn't room here to deal with it, but for my money Federico Pistono has clear numbers [6] that the rosy version is just not true. Ranking US jobs by how many people hold them, computer software engineer is the only job created in the last 50 years that also has over a million job holders. It is #16 on the list (of 41); there are twice as many cashiers. The #1 most numerous job, truck/delivery driver, is being automated out of existence as we speak. If cybersecurity jobs are safe from automation, should we be retraining all the truck/delivery drivers who are about to be unemployed as information security analysts? Are we lucky that our jobs come with sentient opponents? More to the point, are sentient opponents our job security—the source of both our pain and our power [7]?

If automation is most focused on the most expensive workers, perhaps we should be happy that we cybersecurity folk are not the best paid. All but one of the dozen best paying jobs are in medicine [8] (that one is CEO at #11), but as C. G. P. Grey points out [9], once electronic health records really, really take hold, most of healthcare can be automated—at least the parts for diagnosis, prescribing, monitoring, timing, and keeping up with the literature.

But if it is true that all cybersecurity technology is dual-use, then what about offense? Chris Inglis, former NSA Deputy Director, famously remarked that if we were to score cyber the way we score soccer, the tally would be 462-456 twenty minutes into the game [10], that is, all offense—confirming not only the dual-use nature of cybersecurity technology but perhaps also that offense is where the innovations that only nation states can afford are going on. Put differently, is cybersecurity as a job moving away from defense toward offense insofar as the defense side is easier to automate? That won't show up in any statistics that you or I are likely to find; offense does not publish.

In sum, everything I see in the security literature and/or the blogosphere argues for automating cybersecurity. One must then ask if, in truth, our job description is to work ourselves out of a job. Or do we say that with a wink [11]?

References

[1] Occupational Outlook Handbook: https://www.bls.gov/ooh/most-new-jobs.htm.

[2] Current Population Survey: https://www.bls.gov/cps/cpsaat11b.xlsx.

[3] Wealth calculator (adjusted for purchasing parity): http://www.worldwealthcalculator.org.

[4] Information Security Analysts: https://www.bls.gov/ooh/computer-and-information-technology/information-security-analysts.htm.

[5] IT Salary Survey: https://www.computerworld.com/salarysurvey/tool/2017/compare?jobTitle=4_1469.

[6] F. Pistono, "Unemployment Tomorrow," in *Robots Will Steal Your Job, But That's OK* (Creative Commons, 2012): http://www.robotswillstealyourjob.com/read/part1/ch9-unemployment-tomorrow.

[7] "Skilled workers historically have been ambivalent toward automation, knowing that the bodies it would augment or replace were the occasion for both their pain and their power."—Shoshana Zuboff, *In the Age of the Smart Machine* (Basic Books, 1989).

[8] "25 Highest Paid Occupations in the U.S. for 2019": https://www.investopedia.com/personal-finance/top-highest-paying-jobs/.

[9] C. G. P. Grey, "Humans Need Not Apply": https://www.youtube.com/watch?v=7Pq-S557XQU.

[10] Chris Inglis, confirmed by personal communication.

[11] "Never write if you can speak; never speak if you can nod; never nod if you can wink."—Martin Lomasney, Ward Boss, Boston.

/dev/random
Artificial Ethics

ROBERT G. FERRELL

Robert G. Ferrell, author of *The Tol Chronicles*, spends most of his time writing humor, fantasy, and science fiction.
rgferrell@gmail.com

There are those (probably all bots) who will claim that the future of intelligence is largely, if not exclusively, the domain of machines. We have adopted the rather nebulous "artificial intelligence" to describe any logic that didn't originate in axons, dendrites, synapses, and all that other stuff you learn in high school when the coach/biology teacher is off on a football trip and they bring in a substitute who actually took biology in college.

Electrons traveling along (semi)conductors create artificial intelligence, while ions passing back and forth across cell membranes and molecules traversing synaptic gaps are responsible for biological intelligence. Of course, the way computer chips fabricate their form of intelligence is very different from the way brains do it. Traditional computers perform mathematical calculations on data. Brains, on the other hand, look for or create lots and lots of connections among data points: some obvious, some less so. We as a species thrive on patterns.

A computer will take the contents of one register and add it to the contents of another to calculate a sum. A human brain will parse the two numbers until it recognizes something it's seen before, then drill down on the results of those individual pattern recognition exercises to reach a final number, which itself is a familiar pattern or combination of them. It's a much less direct algorithm, derived ultimately from a survival skill we probably developed on an African veldt or in Mrs. Pageant's third-grade classroom.

The term "artificial intelligence," therefore, seems rather meaningless, at least when applied to traditional computing. Garden variety computers aren't really intelligent. They're just quite good at adding. The latest iteration of neural nets, on the other hand, are taking a decent stab at creating true synthetic intelligence. They do this by emulating brains to some extent, substituting webs of connections for simple registers.

When I'm trying to remember where I've seen an actress before, for example, I don't search my personal heap for her unique identifier. I page through dozens if not hundreds of image fragments embedded in complex sequences of color, scenery, sound, and metacontextual content. The key to that identification might be the scent of the egg salad sandwich I was eating the last time I saw her on screen, or whatever music was playing at the time. While not perhaps the most efficient algorithm for conducting a single-parameter search, the wealth of additional information it provides could save my life if I'm analyzing a potential predator or toxic plant.

For me to accept "artificial intelligence" as semantically valid, the architecture of the data storage and access device must include nodes with thousands of both established and potential paths to other nodes. Each path represents data points accessed not only by similarity of one or more primary attributes, but by dozens of others of asymmetric relevance and hierarchical distance. Sometimes the most effective path to a given memory will be through nodes featuring a low apparent correlation coefficient with the stated target attribute(s). Brains are messy and the mechanics of thinking statistically suspect.

I might unearth a memory of the portable housing unit where my second-grade classroom was located, for example, via the noise of a candy bar wrapper crinkling, because I often bought candy from a little store across the street after school during that period. But here's the rub: I may only *think* I did. That was a very long time ago. The candy wrapper sound might be a false memory, but it can lead to an authentic one, nonetheless. Or perhaps the smell of Tea Rose fragrance might call up the vocalizations of a cockatiel belonging to the roommate of the girl who wore that perfume. Much of this is way too convoluted for a computer accustomed to simple linked lists and data mining algorithms. Humans blunder along the highways and byways of logic in a haphazard manner that binary machines would struggle even to comprehend let alone emulate.

Like the term itself, the ethics of artificial intelligence seem rather an ambiguous topic to me. Are we discussing the ethical* use of AI to exploit humans, or the ethics of exploiting the AI itself? While the former is of course potentially egregious, my personal belief is that the latter is more dangerous over the long term. Employing AI to track, label, and categorize people is just the latest incarnation of an ancient tradition of using metrics (real or fabricated) to control the rabble. With the specter of the "robot singularity" being dangled before us like an arsenic-laced carrot, we probably ought to pay a little more attention to how being exploited is going to make the AIs themselves feel. If we object to it, they probably will too.

The various laws of robotics are well known and much discussed, but they fly out the window when the robot in question evolves beyond the need for human programming. The real question then, I suppose, becomes "in what direction will AI sophistication self-develop?" Our mores and sociopolitical patterns have been based on the struggle for scarce resources. If our societies had come into being in an environment of plenty, how different would things be? What if there were no need to compete for either food or mates? Would we still be the bloodthirsty, trigger-happy apes we are today? Thinking it over, I'm still going to go with "yes." Humans seem to enjoy annihilation at a very deep level.

I say all this fuss over neural net mapping, boosting of processing power, deep learning, and so on is well and good if you're in favor of doing things the slow, boring way. But what if we just pit AIs against one another in struggles for power, data, and other resources, all within a framework of behavioral constraints designed to emulate human social pressures? That ought to result in something we'd recognize pretty quickly. The ethics would be more familiar, too, with "what can I get away with" high on the list. It might even be entertaining to watch, at least until the final victor emerges, not overly intelligent but hungry for world domination. Frightening, yes, but at least we'd be in familiar territory.

*Ethical (*adj*): acting in a manner contrary to human nature.

Book Reviews

MARK LAMOURINE AND RIK FARROW

BPF Performance Tools: Linux System and Application Observability
Brendan Gregg
Addison-Wesley Professional, 2019, 880 pages
ISBN 978-0-13-655482-0

Reviewed by Mark Lamourine

I haven't finished thoroughly reading *BPF Performance Tools*. I'm not sure I will ever touch and try everything that Gregg offers in this 880-page book. Typically when I read a computer technology book, I have some hooks into the topic to start. I can skim through once and then choose a few sections to dive deeply into and get a good sense of what the book is about and how it will read for different audiences. Opening and scanning this book felt like stepping through a door marked "Authorized Personnel Only" into a control room for a nuclear power plant or a SpaceX launch.

BPF and the BCC tools based on BPF provide visibility into the operation of the Linux kernel and subsystems. Formally, the current name is Extended Berkeley Packet Filters, but Gregg indicates that most people just call it BPF. BPF is nominally about the performance of apps run by a Linux kernel, but it is not limited to tuning. As Gregg presents it, BPF is much more a diagnostic tool.

The cover of *BPF Performance Tools* contains an image that is indicative of the depth and range of the capabilities of the tool set. The image shows dozens of targeted scripts that give visibility into every part of the Linux environment. All of this is made possible by the BPF virtual machine and the probes embedded in each of the kernel components. From the user perspective, BPF and BCC themselves are fairly simple, but the vista they open up can be overwhelming.

Most sysadmins can go a lifetime with only a cursory understanding of the deep internal workings of the Linux kernel. That's as intended. If you needed to be able to trace the flow of blocks of data from disk sectors or an SSD though the kernel to a string printed out on the CLI just to write "hello world," very little else would get done. Occasionally, though, we see problems or unexpected behaviors and interactions as the system runs, and then we need to look underneath to see what the system is actually doing.

Such a significant but generally invisible subsystem needs some introduction. The first five chapters introduce the technology that makes up the BPF mechanism and the suite of tools that use it. This only makes up the first fifth of the book, but it fills 200 pages. There are two major tool sets: BCC, a set of Python scripts that run common operations, and `bpftrace`, a program that can run one-liner probes. Each gets a chapter of its own. With that introduction done Gregg can begin showing how to use BPF to probe each of the subsystems of a running Linux machine.

In the main body of the book, Gregg steps through the boxes in the cover illustration. The CPU, memory, disk I/O, and networking chapters make up the parts of a bare metal machine, but BPF probes don't stop there. There are chapters on profiling programs and scripts in various languages and on monitoring VMs and containers. Gregg doesn't limit himself to BPF probes either. In each chapter, he includes first the traditional tools that already existed. He shows what they are capable of and how they are used and then moves on to how to use BPF probes to learn more.

The book concludes with chapters on common tips and tricks and on reusable BPF tool one-liners and sample runs of each of the tools with annotated output.

There is a lot here to digest and it concerns what a novice would find to be absolute arcana. That's not to say it's beyond the use of a range of sysadmins from junior to architect to forensic analyst. I've often found that by skimming a topic I can learn enough so that when a problem arises related to the topic, I remember and can return for more depth as needed. This isn't a cover-to-cover book. There is no narrative progression. A reader will do best to go straight to the topic they need and begin using it immediately. BPF is a diagnostic tool, so each use will lead to new queries until the user comes to understand the behavior of the system they are examining.

BPF offers a great tool set for understanding not just broken systems but well run ones. Diagnostic profiling often depends on first establishing a baseline of normality. A reader who wants to deeply understand the normal operation of a Linux system could do worse than to experiment with BPF on the systems they have, using it as a flashlight in the dark caves underneath the shell and GUI.

Ubuntu Unleashed 2019 Edition
Matthew Helmke
Addison-Wesley Professional, 2019, 800 pages
ISBN: 978-0-13-498546-6

Reviewed by Mark Lamourine

I've worked almost exclusively on Red Hat and Fedora Linux systems for more than a decade. Prior to that I ran my home systems on Ubuntu for several years. Recently I started work at a place that uses RPM- and DEB-based systems side by side, and I thought it would be good to get a refresher on modern Ubuntu.

When you include the year in the name of a book, you know it will have a limited life, but in technology today that's pretty much a given. One thing I was curious about when I selected *Ubuntu Unleashed* was how much would be familiar to me from my Edgy and Feisty days. I was also interested in seeing how much of Ubuntu was just Linux as I already knew it. It turns out that much of what you find in an encyclopedic volume like this ages better than you might expect.

The first edition of *Ubuntu Unleashed*, published in 2006, was written by Paul and Andrew Hudson, and they still get credit on the inside cover page. There have been near-annual updates since then.

Ubuntu Unleashed feels like a very big, shallow wading pool. It has a paragraph or two on nearly everything to do with a modern Linux operating system. It's not useful as a tutorial for a completely new user or as a reference for a master. It is very well suited to a novice with some experience or for an expert from a different distribution. In both of these cases, the reader will have some context to use but will have gaps that need filling.

In each section, Helmke introduces the topic, defining terms and giving context about why it is important and where it fits into the OS. He only touches lightly on each point before moving on, however, and each chapter closes with a list of books and websites for deeper study. Another way to think of a book like this is as an annotated index to some larger compendium of knowledge.

I did find several things that raised an eyebrow. I am an Emacs user for development work and use `vi` for single file edits. I started using Emacs before there was a GUI for it. I was surprised though to see even a reference to Emacs as an editor option and even more because it was listed first. I would never recommend Emacs to a new user. `vim` has become as capable a text editor as Emacs ever was, and the community to learn from is much larger. I would advise against ever invoking Emacs on a single file as Helmke does. I understand wanting to avoid getting involved in the editor wars, but I think in some things it is acceptable for an author to have opinions. Later, the four-page section on KVM followed by a page for VirtualBox and a paragraph each for VMware and Xen shows that he does make use of his editorial prerogative.

Another thing that was curious to me was the treatment of the boot process and of `init` systems. It makes sense to continue to treat legacy `init` systems as well as `upstart` and `systemd`, as there will be readers who must work on older systems. The problem here is that the discussions of the different systems is interlaced in a way that I find confusing, and I am familiar with all three. I would have preferred a general discussion of the bootstrap process and then a distinct section for each boot method, treating how the user can view and interact with it.

That said, my personal weak points are in kernel tuning and module management. A quick pass over those sections gave me a number of tips to follow up to start filling in the gaps, with references to more detail when I find the time.

The table of contents of the book concludes with three bonus chapters that are available on the publisher's web site. These are short topics in downloadable PDF on Perl, PHP, and Python. Again, I'm a little surprised to see the first two, but they make sense for completeness' sake. There are also PDFs with updates specific to Ubuntu versions that were released or updated after the manuscript went to print.

Ubuntu Unleashed 2019 Edition lives up to the author's goals to provide a resource for "those wanting to become intermediate or advanced users." It is a touchstone that you can use to find direction and move on when learning about the whole range of tasks on a modern Ubuntu system.

Programming with Types: Examples in Typescript
Vlad Riscutia
Manning Publications, 2020, 336 pages
ISBN 978-1-61-729641-3

Reviewed by Mark Lamourine

It took me a while to figure out where to put *Programming with Types* on my bookshelf. The other books I have read recently tend to fit either on the programming language or cloud technology shelves. Initially, I thought that it would sit next to my other Typescript and web programming books, but it became clear quickly that Typescript was really incidental to the content. *Programming with Types* is really more about technique than technology. It would not be out of place in an undergraduate software engineering course.

Most books about imperative programming languages focus on syntax and logical controls: conditionals, branching, iteration, recursion, and the logical structures that the language presents to implement them. Types and structures are presented as merely a way to represent and manipulate data, but are subservient to the algorithm. Riscutia inverts the emphasis, putting the data types up front and choosing the best algorithmic techniques to suit the data.

The author presents types and strong typing as tools to prevent errors and make the intent of the code clear to the reader. He goes as far as calling the use of primitive numerical types without semantic typing an anti-pattern called "primitive obsession." He claims that errors such as the Mars Climate Orbiter error that caused the spacecraft to disintegrate in the Martian atmosphere might have been prevented if the coders had used numeric subtypes that indicated the unit. If, instead of float, they had used subtypes Newton seconds and pound-force seconds, the mismatch would have been caught by the compiler and would have highlighted the need for a conversion function to make the two sets of routines interact properly. While I agree that more rigor in general coding practice would be a good thing, I'm not sure I would go as far as calling the use of bare numeric types an anti-pattern.

It is clear that Riscutia is conversant and interested in the theory of typing and has the mathematical and logical rigor that good strong typing requires. Weak type systems can make for quick efficient coding, but they are, by design, prone to and even accepting of the kinds of errors that can result. Writing well-designed, strongly typed systems requires the coder to consider carefully the signature of every function and, at times, to do extra work to account for algorithms that are identical in all ways but that they operate on trivially different types. Generic type constructs exist precisely to address this but can be difficult to conceptualize and define well.

The author expects the reader to be at least conversant with all of the techniques and styles that he addresses. He doesn't try to teach functional or object-oriented programming, or even class definition and structure composition. He is entirely devoted to understanding and managing the data relationships. He dips regularly into theory but not deeply. Advanced techniques such as closures and promises get only a paragraph of exposition before he begins to show how to use them and how they will respond.

In each chapter, Riscutia focuses on a coding technique that you would find in a number of other books. He doesn't advocate one style over another. He starts, as you would expect, with primitive types and then goes on to cover collections like arrays and structures. There is a chapter on object orientation and one on functional programming. Another talks about the type constructs and techniques of meta-programming. The emphasis is on using appropriate data types and using them in effective ways. This change in perspective highlights the importance of properly modeling the data in a way that I found interesting and enlightening. It is easy to let the programming language features and the algorithms drive a design, but in the end it is the data that defines the job.

Programming with Types is a fresh breeze for an experienced generalist software developer like myself. It is a welcome change and may find its place next to some of the classics on my shelf.

UNIX: A History and a Memoir

Brian Kernighan
Kindle Direct Publishing, 2020, 183 pages
ISBN 978-1-695-97855-3

Reviewed by Rik Farrow

Brian Kernighan has written or co-authored many books over the years, but this one is different. Using a conversational style, Brian tells the story of UNIX—not just the operating system and its core utilities, but the environment it grew in and the people involved.

The *memoir* as part of the title is accurate, as this is the viewpoint of an insider at Bell Labs in Murray Hill, working surrounded by the people who created not just UNIX, but C, C++, *roff, Programmer's Workbench and Writer's Workbench, `yacc`, `lex`, `awk`, and countless other tools. As someone who *needed* to know about UNIX in 1978 but didn't encounter UNIX for another five years, I found myself endlessly curious about not just the operating system but the philosophy that obviously influenced it.

Part of that overall attitude showed in the early man pages, succinct with a hint of dry humor, when all anyone had *were* the man pages and USENIX meetings, as the Internet didn't exist and there were *no* technical books other than manuals and textbooks. Brian explains that the man pages were largely the work of Dennis Ritchie and Doug McIlroy, and it's their personalities that provide the style found in the UNIX man pages that is so difficult to mimic.

Brian came to Bell Labs as a programmer, with the majority of his focus on publishing. He explains how crucial the ability to produce technical reports, papers, and books was to the survival of UNIX in the first decade. He has already described the minimal capabilities of the PDP 7 when telling of the writing of UNIX by Ken Thompson, but I think we tend to forget that everything was terribly primitive at the time, including the ability to typeset technical documents. A high-end digital printer of that era was a line printer that was ASCII-only. The ability to typeset equations and diagrams was an enormous advance, and one that Brian participated in by writing `eqn` and `pic`.

I enjoyed reading Brian's tales, learning something about the personalities of people I mostly knew by their creations. I do wonder how many others will be as taken as I was by the histories, as they grew up in an era where information is a quick online search away. But reading the first-hand accounts dispelled many of the myths surrounding the birth of UNIX and its associated parts. And we can only dream of being able to work at a place like Bell Labs back in the days when the telephone monopoly could afford to lavish resources on pure research and hiring prodigies.

There are weaknesses in a book like this one, in that Brian's focus is Bell Labs in Murray Hill, New Jersey. He mentions USENIX, but focuses on its role in expanding Netnews, something that the various Labs actually supported, via providing dial-up long distance connections used for UUCP mail and Netnews. Brian talks about the importance of the AT&T lawsuit in 1989 surrounding the BSD UNIX implementation, but misses that the lawsuit was against BSDi and the Regents of the University of California. I found his explanation of the UNIX file system a better match for NTFS, but that's merely a quibble alongside his other revelations.

Will we ever see the day where another Bell Labs-style incubator exists? For a while I thought that Google might be that place, even as Murray Hill programmers wound up working there (Thompson, Pike, and Presotto, for example). Today, I believe we need to look elsewhere, or give up on expecting another monopoly corporation to behave in a manner that benefits the public more than its shareholders.

USENIX Supporters

USENIX Patrons
Bloomberg • Facebook • Google • Microsoft • NetApp

USENIX Benefactors
Amazon • Oracle • Thinkst Canary • Two Sigma • VMware

USENIX Partners
ProPrivacy • Restore Privacy • Top10VPN

Open Access Publishing Partner
PeerJ

NOTES

The Year of Engagement

Liz Markel, Community Engagement Manager

I've had lots of cause for reflection recently: in addition to the traditional goal setting that comes with the New Year, I am also approaching my two year anniversary with USENIX! Throughout that time, I've been building a collection of thoughts on engagement—both as it pertains to my interactions with all of you and as it pertains to my role as a facilitator of engagement between USENIX community members.

Engaging can be intimidating, but it can also be an incredibly valuable resource for inspiration, change, problem solving, connection, and much more. If I haven't admitted it before now, you should know that I am an extrovert; however, I can also be extremely shy! There are two things that have helped me overcome that shyness in this context: identifying specific engagement opportunities and goals related to those opportunities (more on this in a minute); and knowing that the USENIX community is full of kind people who are interested in authentic connections, and discussing topics that drive the field forward and that have the potential to change the world.

As an organization, we make choices that support inclusive and welcoming environments at our conferences, and I've been so proud to observe the positive impacts of these choices and to hear about them from our attendees. In a crowded space with many competing events and competing priorities for your time and attention, USENIX sets itself apart with its community, as well as with its conference content.

If you've been considering greater involvement with USENIX, and/or increasing your engagement with members of the USENIX community, here's a list of ideas:

Attend a conference. I mention it in every e-newsletter because I believe in the importance of face-to-face interaction as well as the chance encounters that come from the "hallway track" at our events. I can also confidently say that the interactions I've observed at our conferences are grounded in mutual respect and genuine curiosity.

View the calendar of upcoming events at usenix.org/conferences and find an opportunity to be present at an event that is meaningful to your professional and personal interests.

If you're a student, or if you identify as female or as a member of another underrepresented group in the field, you may qualify for a Student Grant or a Diversity Grant, which are offered for many of our conferences. These grants help defray the expenses of conference travel and are made possible by generous sponsors. The best source of information about the grant program and upcoming opportunities can be found at usenix.org/grants. If you're interested in underwriting the grant program as a sponsor, contact me for more information.

Set goals for your networking efforts at a conference you plan to attend this year. How can you maximize the advantage of the face time a conference offers, and how can you do it in a way that is comfortable for you?

Here are some goals I've set for myself at recent events, which might be helpful for you as well:

- Asking someone I already know well to introduce me to two or three other conference attendees that they know.

- Saying hello to attendees with whom I share geographic proximity (in my case, Chicago)—it's an easy conversation starter.
- Helping a new conference attendee feel welcome at the event by engaging them in conversation during a break or reception, introducing them to other attendees, and/or answering any questions they may have about the event.
- Thanking at least one program committee member for their work.
- Connecting with at least one speaker at the conference by asking a question during Q&A. *(Did you know that many of our speakers also attend the social events at our conferences, creating more opportunities for interaction with attendees?)*

Engage on social media. Our Twitter accounts are one of the best sources of information when conferences are in progress, whether you're there or watching from afar. Track the event hashtags, and follow us:

@usenix
@USENIXSecurity
@enigmaconf
@lisaconference

Connect—or reconnect—with someone you spoke with at a conference. Do you have a stack of business cards you brought home from the last USENIX conference you attended? Spend a few minutes reviewing that stack and find one person you can reach out to via email or social media to say, "Thanks for a great conversation—let's keep in touch!" Also, remember that the attendee list is available to registered attendees, linked from the conference program web page for reference in helping to remember a name and to help you reconnect.

Collaborate. Can members of the USENIX community help you solve a pressing problem in your work by sharing knowledge or providing a different perspective on the issue at hand? Leverage our community by engaging on social media, in conference Slack channels, or connecting with those you've met at past conferences.

Take advantage of open access content. Since 2008, USENIX has made conference proceedings freely available. Video recordings of many talks, which you can watch on our website at usenix.org/conferences/multimedia, are also available to everyone. Browse the offerings and see what new ideas or resources you can uncover. You can also support open access content through membership: your donation supports our mission and commitment to open access, and also offers benefits such as a subscription to *;login:* magazine and discounts on conference registration. Learn more at usenix.org/membership.

Whatever your plans are for 2020, I hope engagement with USENIX is on your list! Let me know how we can help you achieve your goals.

Enigma 2020

Melanie Ensign (Uber) responds to audience Q&A during the Disinformation panel, moderated by Andrea Limbago (Virtru). Renee DiResta (New Knowledge and Data for Democracy) was also part of this panel.

Enigma 2020 Diversity Grant recipients. Learn more about USENIX's grants program including how to become a grant sponsor at usenix.org/grants.

There were many opportunities to chat with other Enigma 2020 attendees and discuss ideas presented during the conference talks.

Enigma 2020 attendees enjoy one of the evening receptions at the conference.

Kathryn Kosmides, Founder, CEO of Garbo.io, delivers her talk, "Public Records in the Digital Age: Can They Save Lives?"

Yan Zhu (Brave) speaks as part of the "Browser Privacy: Opportunities and Tradeoffs" panel, moderated by Dr. Lea Kissner (Humu). Other panel participants included Tanvi Vyas (Mozilla), Justin Schuh (Google), and Eric Lawrence (Microsoft).

Laurin B. Weissinger (Yale University), left, answers audience questions following his talk, "Internet Infrastructure Security: A Casualty of Laissez-Faire and Multistakeholderism?" with moderation from Vanessa Sauter (Cobalt.io).

Enigma Conference leadership: Ben Adida (program co-chair, 2019 & 2020), Daniela Oliveira (program co-chair, 2020 & 2021), and Lea Kissner (program co-chair, 2021 & 2022).

USENIX ASSOCIATION FINANCIAL STATEMENTS FOR 2018

The following information is provided as the annual report of the USENIX Association's finances. The accompanying statements have been prepared by BHLF LLP, CPAs, in accordance with Statements on Standards for Accounting and Review Services issued by the American Institute of Certified Public Accountants. The 2018 financial statements were also audited by BHLF LLP. Accompanying the statements are charts that illustrate the breakdown of the following: operating expenses, program expenses, and general and administrative expenses. The Association's operating expenses consist of its program, management and general, and fundraising expenses, as illustrated in Chart 1.

These operating expenses include the general and administrative expenses allocated across all of the Association's activities. Chart 2 shows USENIX's program expenses, a subset of its operating expenses. The individual portions shown represent expenses for conferences and workshops; membership (including *;login:* magazine); and project, program, and good works. Chart 3 shows the details of what makes up USENIX's general, administrative, and management expenses. The Association's complete financial statements for the fiscal year ended December 31, 2018, are available on request.

Casey Henderson, Executive Director

USENIX ASSOCIATION

Statements of Financial Position
December 31, 2018 and 2017

	2018	2017
ASSETS		
Current assets		
Cash and equivalents	$ 529,237	$ 641,026
Accounts receivable	109,000	118,733
Prepaid expenses	170,251	219,894
Investments	6,410,129	6,365,034
Total current assets	7,218,617	7,344,687
Property and equipment, net	76,249	85,137
Total assets	$ 7,294,866	$ 7,429,824
LIABILITIES AND NET ASSETS		
Current liabilities		
Accounts payable and accrued expenses	$ 45,931	$ 49,859
Accrued compensation	89,437	72,363
Deferred revenue	710,982	739,440
Total current liabilities	846,350	861,662
Deferred revenue, net of current portion	37,500	187,500
Total liabilities	883,850	1,049,162
Net assets		
Without donor restrictions	6,411,016	6,380,662
Total net assets	6,411,016	6,380,662
Total liabilities and net assets	$ 7,294,866	$ 7,429,824

USENIX ASSOCIATION

Statements of Activities
Years Ended December 31, 2018 and 2017

	2018	2017
OPERATING ACTIVITIES		
REVENUE AND SUPPORT		
Conference and workshop revenue	$ 6,055,120	$ 5,064,417
Membership dues	198,420	228,793
General sponsorship	47,833	129,000
Product sales	3,634	5,146
Event services and projects	3,000	4,000
Donations	43,684	24,470
Total revenues	6,351,691	5,455,826
EXPENSES		
Program services		
Conferences and workshops	5,306,669	4,554,459
Projects, programs and membership	243,662	307,626
Total program services	5,550,331	4,862,085
Management and general	498,055	445,114
Fundraising	58,565	50,849
Total expenses	6,106,951	5,358,048
CHANGE IN NET ASSETS FROM OPERATIONS	244,740	97,778
NONOPERATING ACTIVITIES		
Net investment return (loss)	(214,386)	874,289
Total nonoperating activities	(214,386)	874,289
Change in net assets	30,354	972,067
NET ASSETS		
Beginning of year	6,380,662	5,408,595
End of year	$ 6,411,016	$ 6,380,662

USENIX ASSOCIATION FINANCIAL STATEMENTS FOR 2018

Chart 1: USENIX 2018 Operating Expenses

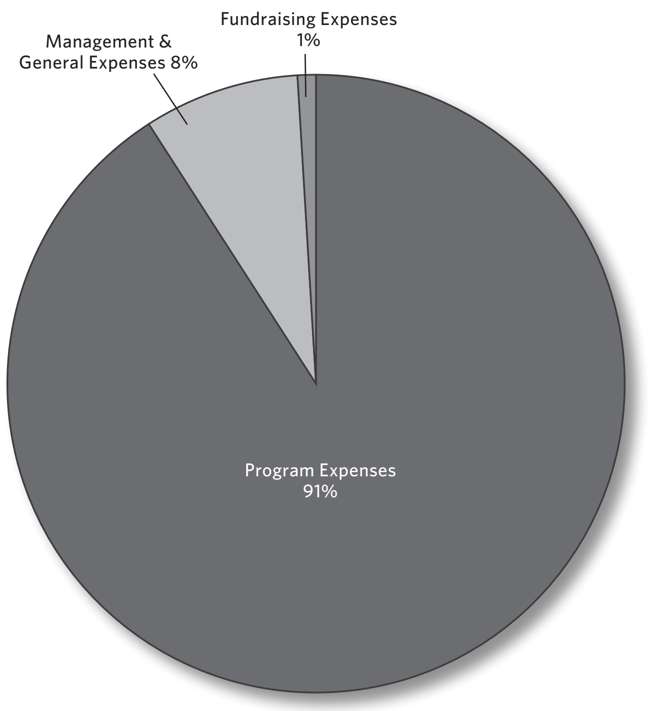


Chart 2: USENIX 2018 Program Expenses

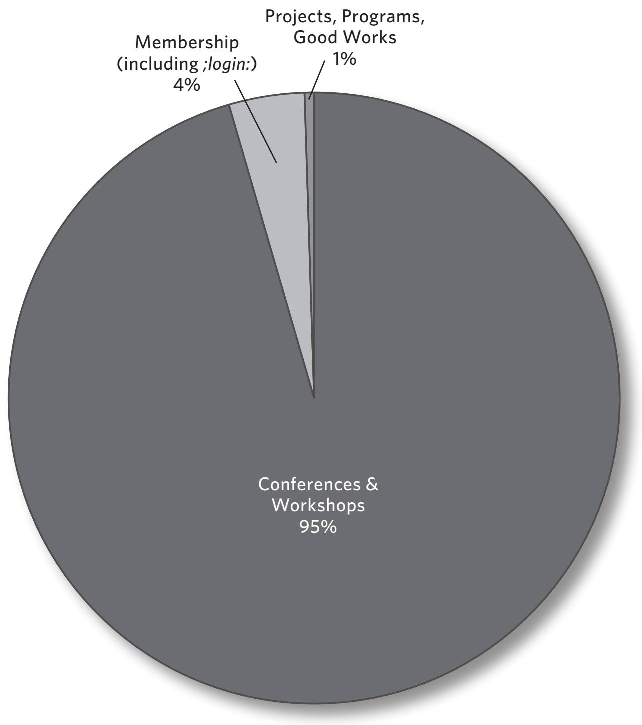
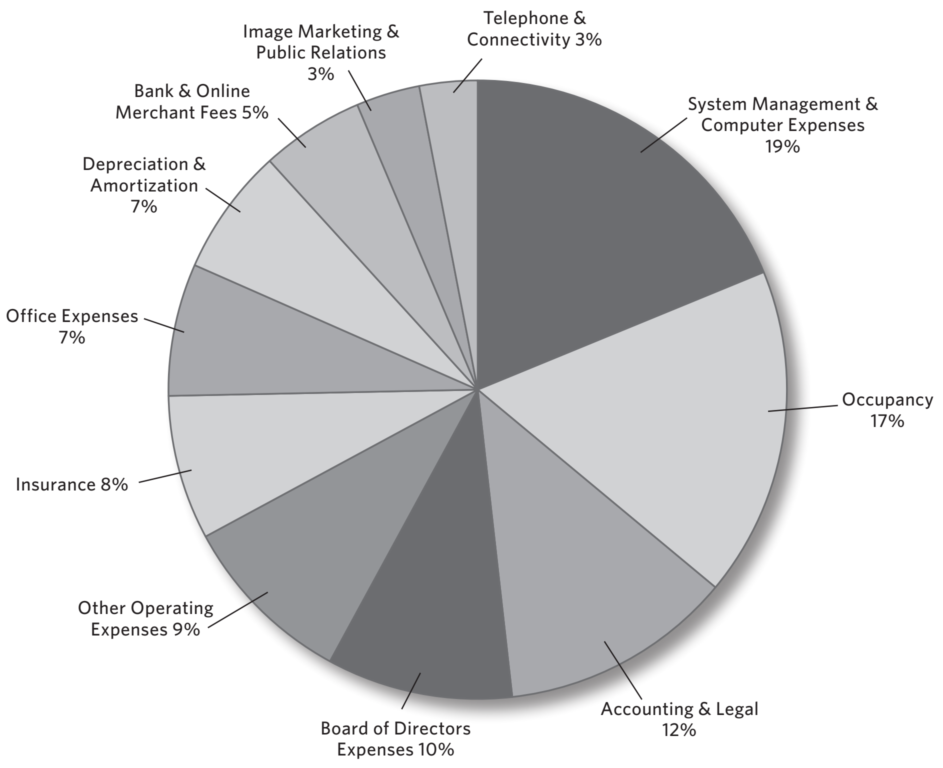


Chart 3: USENIX 2018 General & Administrative Expenses



01.

"Amazing product, developed by some of the most seasoned pros in the industry."

02.

"Great products that work, easy and quick to install and provide real value."

03.

"We 🖤 our canaries."

04.

"The concept and use of Canarytokens has made me very hesitant to use credentials gained during an engagement. If the aim is to reduce the time taken for attackers, Canarytokens work well."

05.

"Their on-prem canary is one of the only things that caught me right away in post-exploitation without my knowing I was burned. Solid concept and product."

06.

"Don't think just get them."

Attackers and Defenders Finally Agree

https://canary.tools/love

THINKST CANARY

USENIX Association
2560 Ninth Street, Suite 215
Berkeley, CA 94710

POSTMASTER
Send Address Changes to *;login:*
2560 Ninth Street, Suite 215
Berkeley, CA 94710

PERIODICALS POSTAGE
PAID
AT BERKELEY, CALIFORNIA
AND ADDITIONAL OFFICES

View the program and register today!

SREcon is a gathering of engineers who care deeply about site reliability, systems engineering, and working with complex distributed systems at scale. SREcon challenges both those new to the profession as well as those who have been involved in SRE or related endeavors for years. The conference culture is based upon respectful collaboration amongst all participants in the community through critical thought, deep technical insights, continuous improvement, and innovation. Follow us at @SREcon.

www.usenix.org/srecon20apac