RH RQ

Bringing great research ideas into open source communities

Kit Murdock

an open source swashbuckler

Fuzzing hypervisor virtual devices

Hardware is back

+

Research Day Europe update

AIOps

Hema Veeradhi on Prometheus Anomaly Detection

Red Hat Research Quarterly
Volume 2:1 | May 2020

Data matters,
where it lives shouldn't.

Query anything anywhere with
Starburst Enterprise Presto
on OpenShift.

TRY TODAY
in the Red Hat Container Library

StarburstData.com

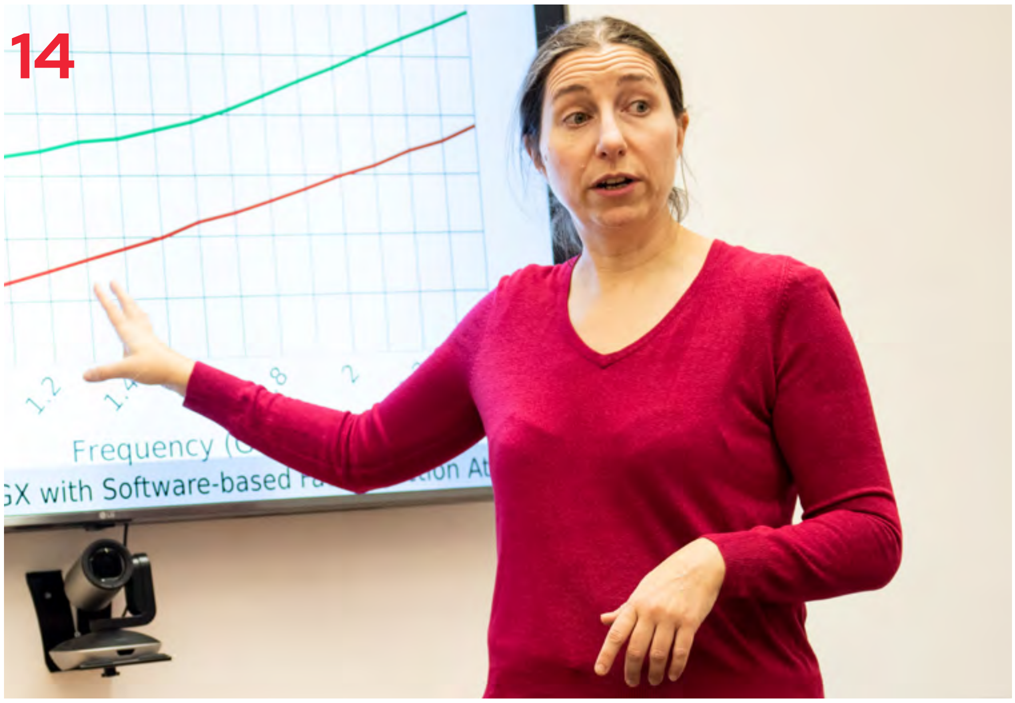
Starburst
ENTERPRISE PRESTO

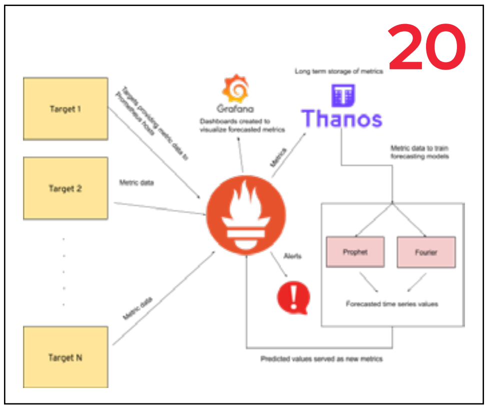
RESEARCH QUARTERLY

VOLUME 2:1

Red Hat

Table of Contents




20
Target 1
Target 2
Target N
Metric data
Grafana
Dashboards created to visualize forecasted metrics
Long term storage of metrics
Thanos
Metric data to train forecasting models
Prophet
Fourier
Alerts
Forecasted time series values
Predicted values served as new metrics

Only connect

In a time when connections are difficult, open source and open science help us maintain our ties even at a distance.

by Hugh Brock

About the Author
Hugh Brock is the Research Director for Red Hat, coordinating Red Hat research and collaboration with universities, governments, and industry worldwide. A Red Hatter since 2002, Hugh brings intimate knowledge of the complex relationship between upstream projects and shippable products to the task of finding research to bring into the open source world.

I have spent a lot of time in the last couple of weeks thinking about the importance of connections. In open source software and in university research, the connections—between people, their work, and the ideas they have in common—are often more important than any single individual. I was really interested, therefore, to read Kit Murdock's thoughts in this issue's interview: "Look at my paper... It's this mass of people sharing ideas, and some people latch onto one and they want to run with it that way. That's what makes great research for me: the collaboration, the ideas, the spin off." Contrast this reality with the myth of the lone developer in a cave, or the solitary genius in his (and myths are usually about *his*) secret laboratory. The power of communication, of willingness to learn from each other, of connection around ideas, is the real engine that drives research as well as open source development.

RH RQ
Bringing great research ideas into open source communities
Kit Murdock
an open source swashbuckler
Fuzzing hypervisor virtual devices
Hardware is back
+
Research Day Europe update
AIOps
Hema Veeradhi on Prometheus Anomaly Detection
Red Hat Research Quarterly Volume 2:1 | May 2020

We have taken this to heart at Red Hat Research, and it accounts for most of the work we do. Take for example our second ever Red Hat Research Day, the first to be held in Europe. We brought together researchers, Red Hat developers, and interested industry folks to hear talks on everything from adaptive learning to side channel attacks. The goal of the day, as for every Research Day, is to connect people with common interests and to influence the direction of research so that it makes an impact. Gordon Haff ably sums up the event in his piece "From dolphins to data to stars." For videos of the talks and the slides we saw that day, see our website: research.redhat.com/research-day-brno. One immediate impact I noticed was a lively exchange of ideas around better ways to identify dolphins from audio recordings using machine learning (for more see *RHRQ* 1:4, "Under the sea: deep learning in marine biology").

One connection becoming more interesting these days is around hardware design. Between open source ISAs like RISC-V, accelerators of all shapes and sizes, improved memory architectures, and more efficient network protocols, we are seeing an explosion of improvements that we hope will combine to produce major gains in compute capacity over time. In his piece "Hardware is back," Uli Drepper lays out the path we expect things to take. Software and hardware vendors will need to connect and cooperate broadly as never before to take advantage of these architectural improvements and make them available to users.

...when we are physically separated, our virtual connections must become stronger, and making them in the open makes them more robust and more lasting.

Finally: As I write this, we in the U.S. are joining the rest of the world in beginning a period of "work from home," of separating ourselves physically from each other to slow the progression of the COVID-19 virus. No one knows how long it will be before things return to "normal," if they ever do, but in a time when people mostly can't see each other in person, virtual connections and virtual teamwork are going to become more and more important and more and more common. Perhaps surprisingly, I think this will be a shining moment for open source software and for research done in the open—when we are physically separated, our virtual connections must become stronger, and making them in the open makes them more robust and more lasting. That is, unless all our videoconferencing melts the Internet...

SAVE THE DATE

09/22/2020
Boston, MA

Red Hat Research Day, US

More information: research.redhat.com/research-day

Dolphins, data, and stars

Real-world applications shine at Red Hat Research Day Europe.

by Gordon Haff

About the Author
Gordon Haff is Technology Evangelist at Red Hat, where he works on emerging technology product strategy, writes about tech trends and their business impact, and is a frequent speaker at customer and industry events. His books include *How Open Source Ate Software* and his podcast, in which he interviews industry experts, is Innovate @ Open.

The phrase "academic research" may evoke thoughts of researchers sequestered in ivory towers, developing theories of what is possible or probable. That work is critical to the development of knowledge. But Red Hat seeks out academics focused on understanding the limitations of the current state of computing and finding practical solutions or different ways of doing things that can have an immediate impact on how we use computers and data.

Red Hat Research Day, held January 23, 2020, in Brno, Czech Republic, brought together members of the global research community who are finding those pragmatic solutions, sometimes by developing new methods and sometimes by combining approaches to provide new results. They covered a lot of ground, ranging from data-intensive science and software to security and privacy to code analysis and verification. All sessions were recorded in full and are available at research.redhat.com. These are just some of the highlights.

OPEN CLOUD TESTBED

Michael Zink of the University of Massachusetts Amherst led off the day with a discussion of the U.S. National Science Foundation-funded Open Cloud Testbed (OCT) project, which integrates testbed capabilities into the Mass Open Cloud (MOC, massopen.cloud), an existing high-performance cloud that allows academic users to run their research projects at no cost.

Using public clouds for certain types of computer science and engineering research also creates challenges. For example, the cloud abstractions over their physical underpinnings can prevent access to data useful for research purposes, such as telemetry data about power consumption. A testbed running in the Massachusetts Green High Performance Computing Center (MGHPCC) allows greater access to low-level hardware and software than is possible with commercial public cloud offerings. OCT will also provide field-programmable gate arrays (FPGAs), which can be used to research configurations such as Bump-in-the-Wire, a method for performing functions such as encrypting data on real-time network links.

MAKING SENSE OF BIG DATA

Gabriel Szász, a graduate student at Masaryk University in Brno, together with a team from Red Hat, talked about their collaborative project using Red Hat® OpenShift® Container Platform for high-performance computing. Szász is studying the effects of rotation on the measured properties of stars, a question in the field of quantitative spectroscopy, one of the cornerstones of modern astrophysics. This study is part of research into the stellar atmosphere model grid.

One of the challenges with this work is that a number of the software components used in the course of spectroscopy calculations are very old and often written in languages like FORTRAN and Ada. But rewriting them would be time consuming, and the rewrites would be subject to acceptance by the broader research community. Red Hat® OpenShift® provides the ability to containerize these components while providing a modern developer experience for new code. OpenShift also provides metrics and dashboards using Prometheus and Grafana. Finally, OpenShift provides the flexibility to run the workloads (and store the 100TB or more of data) on different types of hardware infrastructure as needed.

Georgia Atkinson, a Ph.D. candidate student studying bioacoustics at Newcastle University in the United Kingdom, presented “Acoustic Identification of Cetaceans,” another data-related session. Cetaceans (which include porpoises, dolphins, and toothed whales) are typically top predators in their environment, so their health and numbers tell us a great deal about the health of the marine ecosystem as a whole. One of the techniques for identifying many types of cetaceans is passive acoustic monitoring (PAM), which can distinguish individuals even within the same species by their signature whistles. This talk highlighted the challenges associated with distinguishing these signature whistles among the nine months of ambient sounds collected by three hydrophones.

Techniques included converting the audio to spectrograms that make it easier to detect distinctive frequencies, taking advantage of crowd-sourced dolphin sightings to pinpoint times of particular interest, and filtering samples by the amount of potentially interesting frequency sounds present. It was a useful reminder that real-world data rarely comes clean and ready to use.

You can find the complete program and videos of the presentations at https://research.redhat.com/research-day-brno/

UNDERVOLTING AND PLUNDERING PROCESSORS

During the afternoon's security and privacy track, Kit Murdock, a Ph.D. candidate at the University of Birmingham (United Kingdom), presented "Plundervolt: Pillaging and Plundering SGX with Software-Based Fault Injection Attacks." By taking advantage of a documented interface to dynamically modify processor frequency and voltage, Murdock showed how undervolting a processor could be used to create consistent bit flips. Building on the body of existing research into security exploits that take advantage of bit flips, Murdock demonstrated how undervolting a processor could serve as the basis for RSA, AES, and memory corruption attacks even where Intel's Software Guard Extensions (SGX) were in use.

FORMAL VERIFICATION OF CODE

The final track of the day was dedicated to code analysis and verification. While talks dove into a number of different areas, the primary focus was on formal verification, which can prove correctness of code.

Today, as described by Red Hat's Kamil Dudka, static analyzers are run against the 300 million lines of code and 3,000 RPM packages in Red Hat® Enterprise Linux®. There's a full scan run for each major release and differential scans for each subsequent update. Static analyzers have the advantage of being fast and flexible, but they give both false positives (flagging bugs that aren't there) and false negatives (missing bugs that are, in fact, present). This wastes a lot of time when people are either looking at nonexistent bugs, on the one hand, or potentially shipping software with bugs, on the other.

Red Hat is now experimenting with the formal verifiers Symbiotic and Divine, developed by research groups at Masaryk University. Formal verification doesn't generally replace static analysis, because it's more difficult to use, consumes more resources, doesn't use a predictable amount of compute resources or time, and doesn't deal well with libraries and other dependencies. However, for certain uses, code analysis that doesn't have false positives and negatives while proving code correct is a worthwhile tradeoff. The goal of this research is to augment static analysis with formal verification for important areas of code.

WRAP UP

Red Hat Research Day Europe 2020 highlighted just how relevant fundamental research remains to areas as diverse as systems engineering, high-performance computing, data analysis, and code verification. This research can be enabled by commercial products such as Red Hat® OpenShift® Container Platform, which provide a platform and simplify development for scientists who are not necessarily computer scientists or software developers. For those of us in the field, research into technologies such as formal code verification can help improve the quality of all software products in years to come. RH RQ

Fuzzing hypervisor virtual devices

A security exploit in a popular hypervisor can expose an entire cloud, and all of its users, to theft or worse. Yet hypervisors are notoriously difficult to test because the input space is so large. This joint Red Hat Research/Boston University project takes a novel approach to the problem: use a fuzzer to test all available input combinations.

by Alex Bulekov

Hypervisors—the software that allows a computer to simulate multiple virtual computers—form the backbone of cloud computing. Because they are both ubiquitous and essential, they are security-critical applications that make attractive targets for potential attackers. Past vulnerabilities demonstrate that implementations of virtual devices are the most common site for security bugs in hypervisors. To address this problem, we have developed a novel method for fuzzing virtual devices and implemented it for the popular open source QEMU hypervisor. Our fuzzer combines a standard coverage-guided strategy with further guidance based on hypervisor-specific behaviors. It guarantees reproducible input execution and can, optionally, take advantage of existing virtual device test cases. In our evaluation, we found and reported previously unknown bugs in devices such as serial and virtio-net, ranging from memory corruptions to denial-of-service vulnerabilities. Our evaluation demonstrates that combining well-known coverage-guidance techniques with domain-specific feedback results in promising fuzzer performance, even for complex targets such as hypervisors.

Our fuzzer combines a standard coverage-guided strategy with further guidance based on hypervisor-specific behaviors.

HYPERVISORS AND VIRTUAL DEVICES

Hypervisors, or virtual machine monitors (VMMs), are the cornerstone of cloud computing today. Full-system virtualization offers capabilities such as rapid scaling, live migration, and high availability, without any modifications to the end user's software. Even increasingly popular container-based workloads usually run on top of virtual machines (VMs). Pricing on the cloud is competitive, since multiple customer workloads are collocated on the same bare metal server. Cloud providers and their customers expect the virtualization layer to isolate VMs from each other. In fact, hypervisor isolation is trusted to such a degree that disposable VMs are commonly used for analysis of dangerous malware.

About the Author

Alex Bulekov is an intern at Red Hat interested in Systems Security. He is a Computer Engineering Ph.D. student at Boston University, advised by Professor Manuel Egele.

Hypervisors' key feature—providing a software abstraction over the underlying hardware—is also their Achilles' heel. The software abstraction is composed of meticulously implemented models of devices, which must be compatible with the guest OS' hardware drivers. Device modeling is a complex task, with ample opportunity for errors. Moreover, a single bug in a virtual device can leave millions of VMs exposed to attacks. For example, in 2015 security researchers from CrowdStrike discovered the VENOM vulnerability: a bug in virtual floppy drive code, accessible by default in all x86 VMs running on the QEMU, Xen, and VirtualBox hypervisors. Virtual device implementations like this one are the most common site for VM escape vulnerabilities. Out of the 12 published QEMU CVEs in 2019, 10 were related to vulnerabilities in code accessible through virtual device interfaces.

In the past decade, fuzzing has successfully detected thousands of bugs.

Fuzzing is a technique for dynamically generating and executing randomized test cases. In the past decade, fuzzing has successfully detected thousands of bugs. For example, Syzkaller, a kernel fuzzer, has found over 350 bugs in the Linux kernel alone.

Guided fuzzers leverage information, such as code coverage, to identify randomized inputs that result in new program behaviors. A guided fuzzer identifies these inputs and provides them to a mutation engine, which performs operations such as byte swaps with the expectation that small mutations may uncover more program behaviors.

RESEARCH PROBLEM

Due to the potentially critical nature of virtual device vulnerabilities, fuzzing is a natural direction for proactively fixing bugs in the hypervisor. Unfortunately, in contrast to programs such as image libraries, network servers, and parsers, hypervisors expose a significantly larger input space, which is difficult to target with a standard fuzzer.

Consider the attack surface for VM escape attacks. Since the operating systems running in VMs communicate (or expect to communicate) through peripherals, hypervisors implement virtual devices to simulate them. In the past, virtual devices were commonly based on real devices, such as Cirrus Video or SoundBlaster audio cards, since the real devices already had driver support in mainstream operating systems. Physical hardware faces a different set of constraints than virtual devices, however, so virtual devices often performed suboptimally. As VMs became ubiquitous and inefficiencies resulted in real costs, hypervisor developers designed and implemented paravirtualized devices. Unlike older virtual devices, paravirtualized devices are not based on any physical device. Instead, they are optimized specifically for virtualization. Virtual devices are typically implemented in software with hundreds of thousands of lines of code. Device virtualization is the main interface between virtual machines and the underlying host, and composes a major portion of the attack surface accessible from the guest.

Operating systems typically rely on three major interfaces for communicating with virtual devices:

- **Port-Mapped IO (PMIO)** relies on special x86 instructions (in and out) to access a special 64k-address space mapped to IO device registers.
- **Memory-Mapped IO (MMIO)** devices have their registers mapped into main memory (accessible with standard memory access instructions such as read/write).

- **Direct Memory Access (DMA)** relies on additional hardware to provide virtual devices with direct access to main memory. This results in fast, asynchronous access to large amounts of data, without the need for the OS to copy data over MMIO/PMIO. DMA is used in heavy-bandwidth devices such as network and graphics adapters. DMA-capable devices still rely on PMIO and/or MMIO, since the CPU must communicate the location of DMA data to devices.

In combination, these I/O interfaces expose a virtually limitless input space, while fuzzers perform best with small inputs. Our work aims to reshape the virtual device input space to facilitate fuzzing while maintaining performance and ensuring that inputs are reproducible. We implemented fuzzing for the open source QEMU hypervisor, which is used in systems such as Red Hat® Virtualization, OVirt, OpenStack, Gnome Boxes, libvirt, and proxmox. Much of our work has already been upstreamed, improving the security of QEMU and its derivatives. However, the concepts we discuss are applicable to other hypervisors.

DEVICE-TAILORED FUZZ

Our fuzzing system features two main modes of operation. In the first mode, the fuzzer's input is interpreted by a device-specific stub. For example, to fuzz the virtio network adapter, we wrote a parser (fuzz target) designed to accept randomized inputs and convert them into device IO actions.

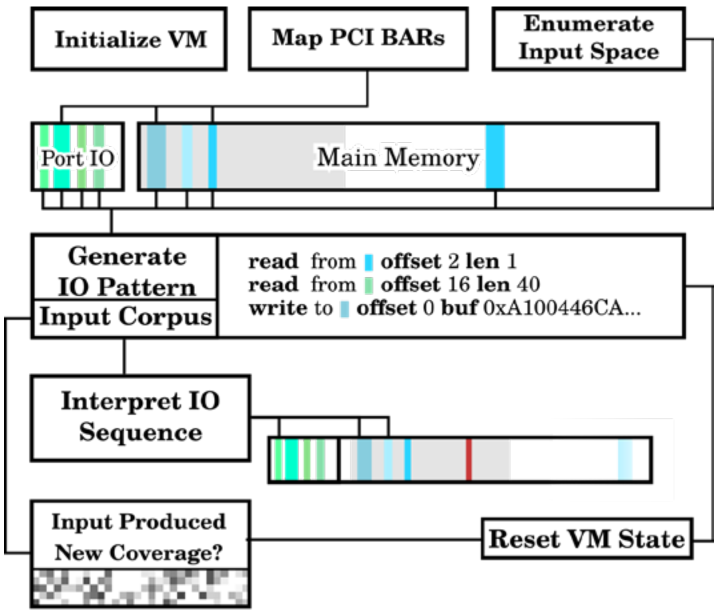


Figure 1. System diagram

The sequence of actions and the data associated with each operation are dictated by the input. The virtio network device exposes queues for data tx/rx and device configuration. Using QEMU's existing testing APIs, we developed a fuzzing target that can result in interactions with each virtio-net IO queue. Manually developing this target allows us to ensure that inputs are handled both safely and correctly. For example, after queuing an input for processing by the virtio-net device, the fuzzer ensures that the input is eventually marked as used.

DEVICE-CONFIGURATION-AGNOSTIC FUZZING

Ideally, we could write tailored fuzzing code for each virtual device. But this would be an overwhelming task: the device-specific expertise required to write such a target, the sheer number of devices, and the possibility of overlooking portions of the input space make this method impractical. Motivated by this problem, we developed our second fuzzing mode: the **configuration-agnostic mode**. The configuration-agnostic fuzzer tests an arbitrary hypervisor configuration, without any device-specific code. Instead, the configuration-agnostic fuzzer relies on generic hypervisor APIs to refine the feedback provided to the mutation engine and increase the chance of triggering new virtual device behaviors. **Figure 1** illustrates how our generic approach initializes the fuzzer, interprets randomized inputs, and leverages feedback from the hypervisor during execution.

Initialization
The generic fuzzer adapts to any virtual device configuration. In a normal VM, the BIOS and bootloader typically enumerate all PCI-capable devices and map their registers to corresponding IO/MMIO regions. Since we wish to interact with these device registers before performing any fuzzing, we perform PCI enumeration to enable all virtual device register-ranges. As mentioned, the Port-IO and RAM address spaces, which contain regions mapped to virtual devices, are expansive. To ensure that reads/writes to the address spaces trigger virtual device behaviors, we leverage the hypervisor's memory API to enumerate all addresses directly mapped to IO device handlers. Using this internal hypervisor API, we locate all enabled PCI devices and devices mapped to hard-coded Port-IO and MMIO locations, obtaining a list of address ranges connected to virtual devices.

Inputs
Similar to the device-tailored fuzz targets, we rely on a parser to convert fuzz inputs to Device I/O actions. To maintain device generality, this parser results in simple read/write operations to Port IO and MMIO locations, rather than interacting with device-specific abstractions, such as queues. By leveraging the IO ranges identified during the initialization stage, we ensure that the fuzzer interacts only with address regions mapped to virtual device registers. Our interpreter is based on a simple opcode grammar. Addresses are represented using tuples of the *index* of the address region, and the *offset* within the address region.

In [Index, Offset] :
Read from a Port IO address

Out [Index, Offset], Value :
Write to a Port IO address

Read [Index, Offset] :
Read from an address in RAM

Write [Index, Offset], Value :
Write to an address in RAM

Memset [Index, Offset], len, buf[] :
Fill len bytes at an address in RAM by repeating the pattern contained in buf[]

DMA buf[] :
DMA reads will be populated with the pattern in buf[]

For individual reads/writes to Port IO/MMIO, the opcode additionally encodes the size of the access (1 byte, 2 bytes, 4 bytes, or 8 bytes). Each fuzz input is parsed into one or more OpCodes. For values with limited ranges, such as opcode numbers or address-range indices, we take the modulo of the value with respect to the maximum possible value of the field. The opcode language can produce arbitrary input sequences to PMIO and MMIO devices. **Figure 2** shows how a random byte sequence is interpreted and executed as a sequence of memory and Port-IO actions. In this example, the "C0DE" byte sequence serves as a separator between actions. As mentioned previously, in addition to PMIO- and MMIO-based communication, devices often rely on direct access data in main memory. Since data can be at arbitrary locations in memory, we cannot isolate possible DMA regions in advance. Instead, we instrument the memory access API to populate DMA buffers, just in time for devices to access them.

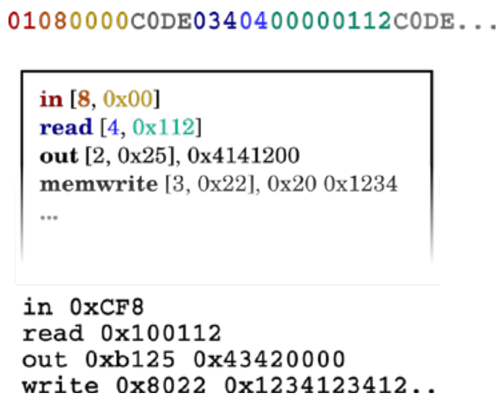

```
01080000C0DE03404000000112C0DE...
```
```
in [8, 0x00]
read [4, 0x112]
out [2, 0x25], 0x4141200
memwrite [3, 0x22], 0x20 0x1234
...
```
```
in 0xCF8
read 0x100112
out 0xb125 0x43420000
write 0x8022 0x1234123412..
```

Figure 2. Input example

Feedback
We base our fuzzer on the libfuzzer library. By default, libfuzzer relies on coverage feedback to guide input mutations. Unfortunately, this feedback alone is often insufficient. For example, DMA-capable devices often treat values written to Port IO and MMIO registers as RAM addresses. On a system with a 64-bit address space, there is a very small chance that a random 64-bit value corresponds to a valid physical address, since 64-bit systems can theoretically support exabytes of physical memory. To address this problem, we instrument QEMU's memory access API to

compare memory addresses against the physical memory limit. Inputs triggering access to addresses within or close to the small physical address-space range result in positive coverage feedback, and are prioritized by the fuzzer's mutation engine.

PERFORMANCE AND STABILITY

The success of our fuzzer hinges on its ability to rapidly mutate and execute inputs. Additionally, the fuzzer should reset device state to ensure identical behaviors for multiple executions of the same input, to ensure that input mutation is effective. Though these requirements are not unique to hypervisor fuzzing, the memory requirements and startup overhead over virtual machines exacerbates the problem. We first attempted to reset the virtual device state by rebooting the guest VM. Unfortunately, rebooting leads to expensive performance overheads. We encountered the same issue when relying on QEMU snapshotting features to rollback VM state after each run.

Instead of relying on a built-in QEMU feature to reset device state, we implemented a fork-server mechanism to guarantee that any virtual device changes do not leak between runs. The fuzzer starts the guest and initializes PCI devices in a single "server" process. After the heavy initialization is complete, the parent process generates and mutates inputs. Rather than executing them directly, however, it uses the fork() system call to create a duplicate QEMU process to safely execute the generated input without affecting any of the server process state. To report coverage information to the mutation engine, we mark the memory containing coverage bitmaps as "shared." This approach not only avoids expensive initialization/cleanup for each run, but also guarantees that inputs do not leak state, and are reproducible.

FUZZING IN PRACTICE

We implemented tailored fuzzers for QEMU virtual devices including virtio-net, virtio-scsi, megasas, and i440x. We also implemented the device-agnostic fuzzer, which enabled us to rapidly fuzz over 70 configurations of QEMU devices. We have already reported bugs in QEMU's serial console, virtio-net, virtio-blk, virtio-scsi, MSI-X implementation, many of which have already been fixed. Our testing identified several classes of bugs, including null-pointer dereferences, stack-overflows, heap-overflows, and assertion failures. We are developing tooling to report bugs to developers efficiently, along with instructions for reproducing them.

As a result of our work, QEMU users are already safer from VM escape attacks. In the future, we will continue to explore ways to leverage the hypervisor abstraction to facilitate virtual device fuzzing. We are also exploring ways to leverage our system to fuzz data entering VMs, such as incoming network traffic. Implementing QEMU fuzzing on the OSS-Fuzz platform, now in process, will enable long-term continuous fuzzing of existing QEMU code and the development of new patches.

The success of our fuzzer hinges on its ability to rapidly mutate and execute inputs.

Interview

Open source swashbuckling

Kit Murdock's team approach to cracking cybersecurity puzzles

by Lily Sturmann

About the Author

Lily Sturmann joined Red Hat in 2019 after obtaining her master's degree concentrating in software engineering from the Harvard Extension School. She works as a software engineer on emerging technologies using hardware-based encryption to improve the runtime security of applications in the cloud, specifically the Enarx project. She is an open source enthusiast and believes that the users of technology should have the opportunity to be the shapers of technology as well.

We first met Kit Murdock at her fascinating talk on the Plundervolt exploit at Red Hat Research Day Europe. It quickly became clear we had to learn more, both about her story and about what attracted her to cybersecurity research. We asked a newly minted member of our own cybersecurity team, Lily Sturmann, to interview Kit. We think you'll find the answers as intriguing as the now-famous Plundervolt logo she inspired.

Lily Sturmann: I want to talk a little bit about Plundervolt to start, which is some very exciting new research that you presented at Red Hat Research Day Europe, in Brno, Czech Republic, in January 2020. Could you give a three-sentence summary of what Plundervolt is about?

Kit Murdock: Plundervolt is an attack on Intel SGX enclaves, which are trusted execution environments. We showed that we can get secrets out of SGX enclaves by lowering the CPU voltage while it is performing calculations. Plundervolt is a problem because SGX has an attacker model that says, "Even if you're root, you should not be able to look inside my encrypted area."

Plundervolt came out of another attack called CLKscrew, which was the first of its kind. Which is kind of cool in cybersecurity, to do a first of its kind! Normally when people talk about hardware faulting, they think of an oscilloscope and some wires, and you're having to take things apart. The fact that you could modify something in software and it would have a hardware impact is kind of a new attack.

It was from that attack, which happened on ARM, that we moved over to Intel and said, "Well in this attack, they attacked frequency. Maybe we can attack voltage." It was because of the granularity: with frequency you can have 2.8 gigahertz, 2.6 gigahertz, but there's nothing in between. It's very hard to change frequency, whereas voltage you could drop millivolt by millivolt. So we tried setting frequency to be one thing and dropping the voltage a very small amount. We found we could get faults and we didn't have to open anything up and we didn't have to get an oscilloscope out.

Lily Sturmann: That's a much more powerful type of vulnerability, and it's much harder to fix as far as I understand it, because the attacker doesn't have to be there on premises manipulating anything.

Kit Murdock: Yeah. And if you're an employee of a cloud provider and you've got access to machines, you can create hardware faults on them.

Lily Sturmann: Your attack has this great catchy name as well. Can you tell us a little bit about the naming process?

Kit Murdock: So when we actually submitted the paper, it was called Undervolt. But actually

we'd come up with a few names, and we left them all in the paper, sort of hidden in the comments. I was a little bit sneaky in that I got my partner to mock up a few logos, pirate-themed logos, because I liked the name Plundervolt. I was the most junior person on the paper, but when [the other authors] saw the pirate-themed logo, they jumped on board.

And the thing that became interesting was that logo, the Plundervolt logo. Like if you Googled it, the only thing you could see was people taking the logo and putting it on top of chips. It caught a lot of attention, which in retrospect is brilliant.

Lily Sturmann: And that must also, in a smaller way than the research itself, feel validating to have that as a cool idea that turns out to be right?

Kit Murdock: Yeah. Because I'm new to the field but have a lot of life experience, there is a little bit of a weird dynamic in that I want to say, "Actually I think I'm right," but on the other hand, my supervisors have been in academia for 20 years so

Kit Murdock is currently pursuing a Ph.D. in cybersecurity at the University of Birmingham. Her research interests include embedded hardware- and software-based fault injections. Kit has been building and researching a tool to enable testing and evaluation of hardware fault injection using software emulation. Kit currently runs the University's Ethical Hacking Club, AFNOM (A Finite Number of Monkeys), which encourages students to learn offensive security in a friendly, informal environment.

it's very hard to say, "No you're wrong." It also is a balancing act, because on the one hand we want our work out there and we want people to know about it, but we also want to be taken seriously and you want to get more research grants.

Lily Sturmann: I wanted to ask you about the impact of open source on your project. Does open source relate to your research particularly, and if so how?

Kit Murdock: So we use a whole host of open source tools. First, we built upon somebody else's research. And it turns out we weren't the only people who had the idea, because there have been other very similar things that have come after us. The second thing is we wanted to be able to do what the CLKscrew researchers were doing: we wanted to modify in software something that changed the hardware and we didn't know how to do that. Intel actually didn't document how to do it, but people on the Internet wanted to know how to overvolt and undervolt their computers, and they had found a model-specific register that you could modify in software. They'd reverse engineered it and they'd put it out there for everyone to share, to see, and to use.

Then I came to look at some tools for overclocking and my mind was blown. There's probably 20 or 30 tools, lots of people who are all creating tools for other people to use that they just give away mostly. So we relied on people just doing something because they wanted to and sharing the knowledge.

Then, when we'd formed some attacks, we just went to Github and typed in Lenstra attack, got a Python script, and we found the vulnerability within a few hours, because we're using somebody else's script to actually read the data and produce the outcome.

Lily Sturmann: You mentioned that you all worked with Intel a little bit to actually disclose this. I'm curious about what that process was like, because Intel has an interest in disclosing this type of information in a specific way. What was their response and how was it to work with them?

Kit Murdock: They were great. So we sent them a proof of concept and they came back not long after saying, "Yes, we've evaluated it, and, yes, we acknowledge that this is a real thing." And then we found a few more attacks, which we also sent to them. We told them we were submitting a paper and what the deadlines were and when we'd hear it, and they were really interested in the whole thing. I think they were really good to us. They were really keen to know what was going on.

The thing I'd like to know is did they know more about the vulnerability than us? So after we disclosed it, I'd love to know whether they went and had a look under the hood and went, "Oh no, this is terrible." At no point did they come back and go, "Oh by the way, did you know you can also do this, this, this, this, and this."

Lily Sturmann: Yeah, that's too bad because I'd also be curious.

Kit Murdock: We actually don't know the underlying problem. We can make some educated guesses and hypothesize; I think we can be fairly sure. But I wonder if there is someone in Intel who knows this exact problem and you can reproduce it like so.

Lily Sturmann: Do you have a hypothesis that you want to share?

Kit Murdock: Well, it's just that the operations aren't completing before the next clock ticks over. So there's a clock tick and whatever's in that register just gets taken and used. If the register is inaccurate because the computation hasn't finished, it's got flipped bits, but obviously that's about 5% of what's really going on. And it's very hard to flip the very low bits. Like we couldn't get one to flip or two or four. So it'd be interesting to know, is there a way to get them to flip or is that never going to happen?

Lily Sturmann: I remember it was also the more complex operations where this was easier to manipulate. You had said in the paper, it's probably because those take longer to get around the circuitry because there's more going on.

Kit Murdock: Yeah, they are aggressively optimized, I think is the phrasing we use. They just fit within the number of clock cycles. So, if they waver by a little bit, suddenly the answer is wrong.

Lily Sturmann: You worked on a fairly large team doing this research, which I think relates to open source because it's a collaborative approach to working. Tell me a little about that.

Kit Murdock: I'm going to talk a little bit about something that sometimes annoys me, which is when you go for a job interview or something, people often value super technical skills. But sometimes it is more important to have the skill of working with people and knowing that you're going to have to accept that somebody else may know the right thing and it may be different to the way you feel about it. I feel that sometimes these skills aren't appreciated as much as, "Oh, you can hack for four hours straight without needing a cup of tea." I don't know that four hours straight without a drink is productive. Maybe two blocks of two hours with a cup of tea in the middle is more productive.

That's what makes great research for me: the collaboration, the ideas, the spin off.

Lily Sturmann: Definitely.

Kit Murdock: I mean, look at my paper, it has six names on it. This is not a single person sitting in a dark room for 40 hours straight. This is people talking and having ideas and going, "I don't think so, but try it," and then going, "Actually, you were right." It's this mass of people sharing ideas, and some people latch onto one and they want to run with it that way. That's what makes great research for me: the collaboration, the ideas, the spin off.

Lily Sturmann: I want to ask a little bit about how you got into the field. You said earlier that one of the qualities you think is most determinant of someone's success is actually resilience, which I couldn't agree more with. So I was hoping you could say a little about your experiences with that, just being new in the field and what that experience is like.

Kit Murdock: I did a career change—I used to write help desk software. I could see the writing on the wall because a lot of work was being taken from the U.K. and sent offshore. I could see that my days were numbered in terms of having an income. So I decided to do a career change and I became a maths teacher in a secondary school. I didn't last very long because I didn't like it. But actually, it's weird, because it almost feels now like a natural progression.

Maths is one of those subjects that people always say, "Oh I'm no good at maths. Oh, never been any good at that. I just don't get maths." And what the research says about teaching kids maths is that we have to get away from telling a child, "You're good at maths. You're bad at maths." We have to talk about the effort that's given, so, "I can see you worked hard." And when someone fails, don't say, "You failed," say, "You're not there yet. It's going to require a bit more work."

One of the indicators of success in children in relation to maths is what they termed resilience. When a problem is difficult, how soon does the child push the paper away and go, "I can't do it," or, "It's too hard," or, "I'm no good at maths." We have to change their mindset to say, "Okay, I need to work harder," or, "I haven't got it yet." So coming into cybersecurity, it's really weird that this word resilience keeps coming up within me. I worked for six months on somewhere that goes, "No," but I didn't throw my toys out of the pram and go, "I'm quitting academia. I didn't get any results." And I'm seeing it in the people who surround me who are successful—that resilience, the ability to go, "Not yet. I haven't got it yet. I'm going to try something else."

I think resilience in cybersecurity goes hand in hand with creativity, because you can't keep doing the same thing and expecting something different to happen. But if you can go, "Oh, what can I try creatively? I'm going to try something else. Okay, that didn't work. What would be a different way of approaching the problem?" So for me, resilience is the single most important thing you can have in computer science and academia.

Lily Sturmann: I know you run a hacking club in your area. I really wanted to ask about it because it sounds awesome.

Kit Murdock: So it's brilliant. I love it. I did a master's in cybersecurity and I came along to it, and then I came back and did my Ph.D. and now I and the person who ran it before me,

Kit Murdock (fifth from left) leads a hacking club at the University of Birmingham.

Andrea, run it together. She's just finished her Ph.D. We've got some of the undergraduates to come and do occasional stuff and they help us, but we're the face of it, we're the hacking club. And we entered a national competition maybe a couple of months ago. It was all online, and they had a prize for the university hacking club that had the most women that day out of that competition, and we won that prize.

I really feel that having two women at the front of it helps people walk through the door and go, "Yes, this is for me." I think it also matters to me and Andrea because we are the only two women in the cybersecurity department. It matters to us that women feel included. Have you heard of the Bechdel Test?

Lily Sturmann: Yes. It's whether two women in a movie talk to each other about something that's not a man. I've been keeping an eye on that kind of stuff for a long time as well.

Kit Murdock: If you get something that's vaguely technical, the woman [in the film] tends to be the slightly ditzy one who goes, "Oh, can you explain it to me? I didn't get it." And so for me and Andrea, it was really important that everyone walks through the door and goes, "Okay, this can be for me."

Lily Sturmann: Yeah, absolutely. It drives me crazy when I see things like that, because speaking of resilience, I think when people don't see themselves in a technical field, they're more likely to give up. And they're more likely to attribute it not to the fact that they haven't tried for long enough and instead just say, "Oh, I must be secretly that ditzy person from those movies. And I just didn't know it yet." It's really sad.

Kit Murdock: So back to hacking club. For the first term of each year, we do led sessions. We did one on networking, we did one on wifi hacking, we did one on reverse engineering, we did one on website hacking. And then in the second term we start actually using the skills. There is a great website called Hack The Box, and we attack those. And then we'd go to university competitions, or we do CTFs that are on the weekend. So there we're a university team competing against any CTF team in the world. I think we're the third-place British university.

Kit Murdock (right) with her fellow hacking club leader. "I really feel that having two women at the front of it helps people walk through the door and go, 'Yes, this is for me.'"

Lily Sturmann: That sounds really fun.

Kit Murdock: I love it. I just went to a girls' school a couple of weeks ago to do a careers day. It's really hard to explain how much fun it is trying to solve a problem. Imagine you have fun with crosswords or Sudoku. It's the same thing, and you get paid for it. This could be your career to do and solve puzzles. Why would you not want to do that? RH RQ

You can find a link to Kit Murdock's Red Hat Research Day recorded talk at research.redhat.com/research-day-brno/

AIOps: Prometheus anomaly detection

Separating the signals from the noise in modern cloud applications is becoming very difficult for human operators, even those armed with experience and copious scripts. Here, we describe a machine learning model to detect the anomalies that matter—how to create it, and what to do next.

by Hema Veeradhi

About the Author

After earning her Computer Science master's degree from Boston University, **Hema Veeradhi** joined the Red Hat AI Center of Excellence team as a software engineer in the office of the CTO, where she explores and integrates open source AI operations across Red Hat platforms.

With an increase in the number of applications deployed on Kubernetes, still a relatively new platform, there is a strong need for application monitoring. Most of these applications are monitored via open source Prometheus metrics, resulting in the accumulation of a large number of time series metrics. Some of these metrics are anomalous in nature, and it is difficult to identify them manually. To address this issue, we at Red Hat's AI Center of Excellence (AI CoE) came up with an AI-based approach of training a machine learning model on these metrics to help detect anomalies.

AIOPS APPROACH AND IMPLEMENTATION

With the increased amount of Prometheus metrics flowing in, it is getting harder for system operators and performance engineers to see the signals within the noise. The current state of the art is to graph out metrics on dashboards and alert on thresholds, which is currently done by domain knowledge, i.e. the people that know the system.

Through an AI-based approach, we can train machine learning models on historic metric data to perform time series forecasting. The true metric values can then be compared with the model predictions. If the predicted value differs from the true metric value, we can report this as anomalous behavior.

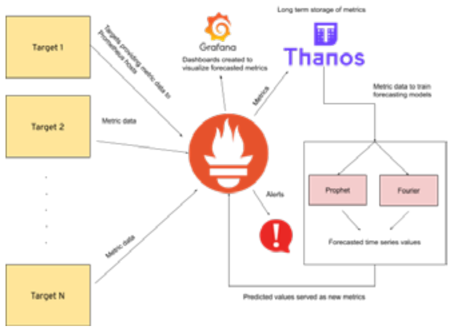

Target 1
Target 2
Target N
Targets providing metric data to Prometheus hosts
Metric data
Metric data
Grafana
Dashboards created to visualize forecasted metrics
Long term storage of metrics
Thanos
Metrics
Metric data to train forecasting models
Prophet
Fourier
Forecasted time series values
Alerts
Predicted values served as new metrics

Figure 1. *Anomaly detection process*

The current implementation of anomaly detection (github.com/AICoE/prometheus-anomaly-detector) within the Open Data Hub project (opendatahub.io) running on the Kubernetes-based Red Hat® OpenShift® Container Platform is shown in **Figure 1**.

1. **Data**: Prometheus time series metrics scraped from specified hosts/targets

2. Models being trained:

a. Fourier: Fourier is used to map signals from the time domain to the frequency domain. It represents periodic time series data as a sum of sinusoidal components (sine and cosine).

b. Prophet: The Prophet (facebook.github.io/prophet/) model was developed by Facebook (and contributed to open source) for forecasting time series data. It is based on an additive model where non-linear trends are fit with yearly, weekly, and daily seasonality, plus holiday effects. Prophet works best with time series that have strong seasonal effects and several seasons of historical data. The following are the forecasted values calculated by the model:

- yhat - Predicted time series value
- yhat_lower - Lower bound of the uncertainty interval
- yhat_upper - Upper bound of the uncertainty interval

3. **Visualization:** We can visualize the time series metric behavior by creating graphs in Grafana. **Figure 2** shows how we can plot and compare the actual and predicted metric values on a dashboard.

This dashboard plots the current metric value, predicted metric value, upper and lower bounds for the predicted value, and the anomaly detected.

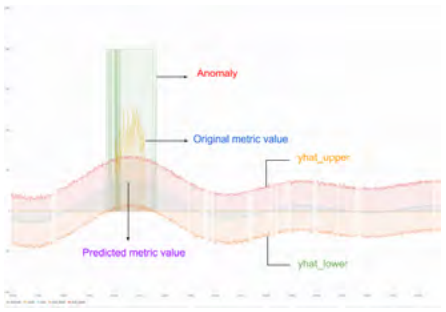


Figure 2. *Anomaly detection dashboard*

4. **Alerting:** The anomalies detected are sent out as alerts using Prometheus. The alerts are configured to send automated messages through a chatbot via Google Chat and email, notifying the respective teams. The Google Chat alerting architecture is maintained in two parts:

   - The predicted metrics are scraped by Prometheus running in the Open Data Hub.
   - Open Data Hub sends a webhook trigger to team Thoth Bot Sesheta, which is configured to trigger alert notifications in Google chat.

The Prometheus instance can be configured with alerting rules for the anomaly detection metrics. Alerting rules allow you to define alert conditions based on expression using Prometheus expression language and to send notifications about firing alerts to an external service. Whenever the alert expression results in one or more vector elements at a given point in time, the alert counts as active for these elements' label sets. **Figure 3** is an example of how an alerting rule can be used to trigger the alerts on anomalies detected.

```
alert: Dgraph Read Failures
Fourier Zero-0 (stage)

expr: (badger_disk_reads:rate1m_
Fourier{ae_source="http://prometheus-
exporter-zero-0.thoth-dgraph-stage.
svc:8080/debug/prometheus_
metrics",instance="prometheus-
anomaly-detector-fourier.cloud.
paas.psi.redhat.com:80",job="Thoth
Dgraph Anomaly Detector Fourier
(stage)",monitor="datahub",value_
type="anomaly"}==1)

for: 10m

annotations:

  summary: "Dgraph Read Failures
Fourier Zero-0 (stage)"

  severity: "MEDIUM"
```

Figure 3. *Automated trigger for alerting rule*

Here, the Dgraph badger disk read metric with the label `value_type="anomaly"` is determined as anomalous if the value is set to 1 or non-anomalous if it is 0.

USE CASE

Project Thoth (thoth-station.ninja/) is part of the AI CoE's AI-driven

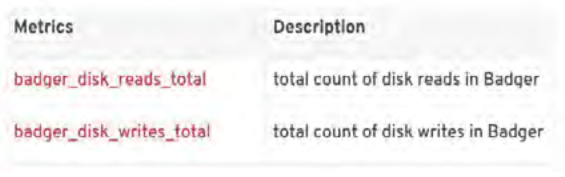
DevSecOps efforts. Thoth is a recommendation system for AI and machine learning applications that uses popular open source machine learning libraries like TensorFlow or PyTorch. Thoth stores software stacks, observations, and information about runtime and build time environments in a graph database. Using the observations stored in the knowledge graph, the Thoth adviser provides recommendations on efficient software stacks for the AI and machine learning applications. Recommendations may include performance-relevant information, CVEs, or other information derived from build time observations.

One of the databases used by Project Thoth was Dgraph. Dgraph (docs.dgraph.io/) is an open source, scalable, distributed, highly available, and fast graph database, designed from the ground up to be run in production. Project Thoth used Dgraph extensively, with all the operations triggered on a per-second basis. As the Dgraph database was an integral part of the Project Thoth architecture, monitoring it was critically important.

This method of anomaly detection was helpful for Team Thoth for monitoring and maintaining the Dgraph database instance.

Dgraph was deployed on an OpenShift instance where it used persistent storage to commit the data and store it. We monitored the read and write actions with Prometheus metrics. The Dgraph Instance provides metrics that follow Prometheus standards. It exposes metrics via the **/debug/vars** endpoint in JSON format and the **/debug/prometheus_metrics** endpoint in Prometheus's text-based format. The Dgraph database doesn't store the metrics, and it only exposes the value of the metrics at that instant. The Dgraph used by team Thoth was deployed with the Prometheus exporter to export the metrics of Dgraph.

The disk metrics track the disk activity of the Dgraph process. Dgraph does not interact



Metrics	Description
badger_disk_reads_total	total count of disk reads in Badger
badger_disk_writes_total	total count of disk writes in Badger

Figure 4. *Anomaly detection process*

directly with the filesystem. Instead, it relies on Badger to read from and write to disk. These metrics can be used for monitoring the read and write to persistent storage.

Project Thoth wanted to have real-time as well as future prediction of the read-write failure of the database. For the future prediction, we needed anomaly detection and action recommendations.

INSIGHTS AND OBSERVATIONS

The anomaly detection in the Thoth Dgraph metrics was valid in the sense that whenever an anomaly was predicted, it was observed that the Dgraph database was indeed failing. When the system detected an anomaly, the Thoth team received email and Google chat notifications for each alert.

One of the incidents we discovered was the host cluster being down, causing the Dgraph read/write failures. During this downtime, the Prometheus anomaly detection triggered alerts for the Dgraph read/write failures, as shown in **Figure 5** (opposite page).

Figure 6 (opposite page) shows some notified alerts sent to Thoth station Google Chat.

CONCLUSION

This method of anomaly detection was helpful for Team Thoth for monitoring and maintaining the Dgraph database instance. It was also helpful in understanding the limitations and hardships of

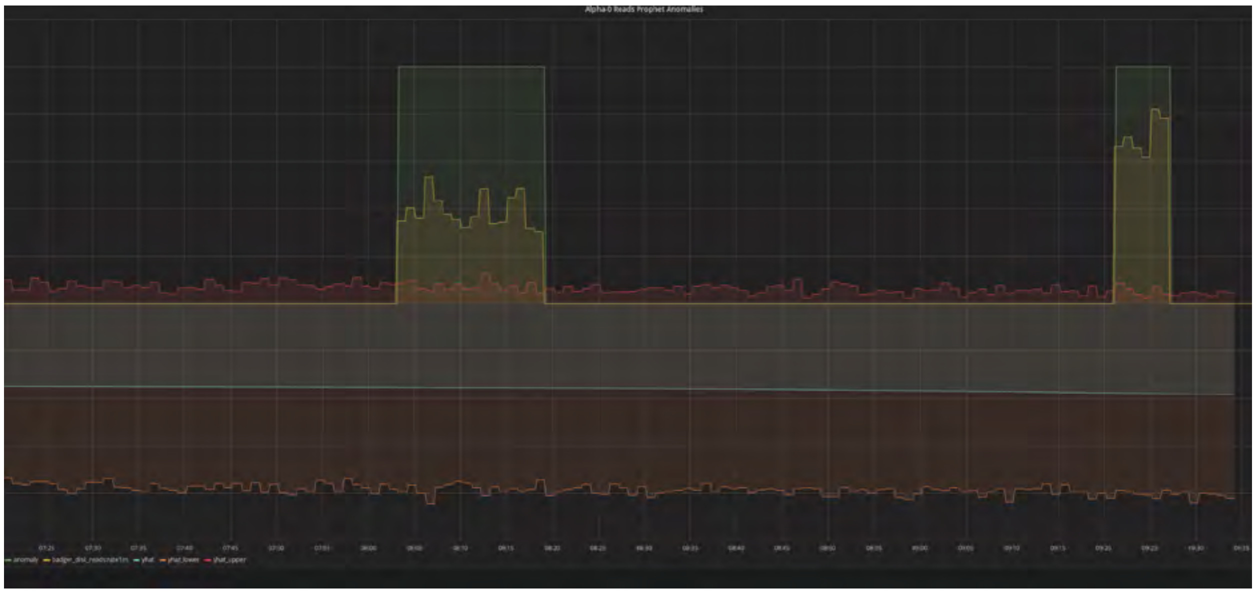
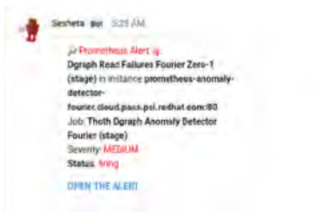


Figure 5. *Grafana dashboard: anomaly detection during cluster downtime*

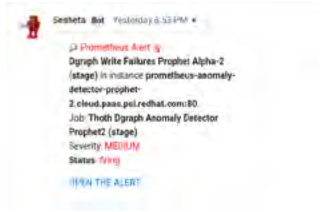
maintaining the Dgraph Instance on an OpenShift cluster. From an AIOps perspective, we were able to accurately alert on the anomalies without any predetermined thresholds. Working with the Thoth Dgraph metrics helped us improve our existing machine learning models. It also led us to consider different possibilities of testing this framework for a wider range of data sets, not limiting its use to a specific metric type. Future work for the Prometheus anomaly detection would be to apply the framework to different platforms and use cases. RH RQ

Acknowledgements:
Thanks to everyone who helped in any possible way.

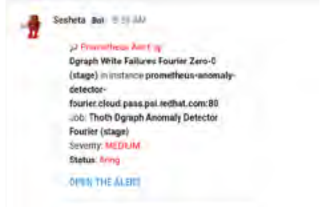
Red Hat AIOps Team: Marcel Hild, Anand Sanmukhani, Michael Clifford, Hema Veeradhi
Red Hat Thoth Team: Christoph Goern, Francesco Murdaca, Frido Pokorny, Harshad Nalla
Red Hat Open Data Hub Team:
Alex Corvin, Maulik Shah


Sesheta Bot 5:25 AM
Prometheus Alert
Dgraph Read Failures Fourier Zero-1 (stage) in instance prometheus-anomaly-detector-fourier.cloud.paas.psi.redhat.com:80
Job: Thoth Dgraph Anomaly Detector Fourier (stage)
Severity: MEDIUM
Status: firing
OPEN THE ALERT

Dgraph zero Read Failure based upon Fourier Prediction


Sesheta Bot Yesterday 6:53 PM
Prometheus Alert
Dgraph Write Failures Prophet Alpha-2 (stage) in instance prometheus-anomaly-detector-prophet-2.cloud.paas.psi.redhat.com:80
Job: Thoth Dgraph Anomaly Detector Prophet2 (stage)
Severity: MEDIUM
Status: firing
OPEN THE ALERT

Dgraph alpha Write Failure based upon Prophet Prediction


Sesheta Bot [illegible]
Prometheus Alert
Dgraph Write Failures Fourier Zero-0 (stage) in instance prometheus-anomaly-detector-fourier.cloud.paas.psi.redhat.com:80
Job: Thoth Dgraph Anomaly Detector Fourier (stage)
Severity: MEDIUM
Status: firing
OPEN THE ALERT

Dgraph zero Write Failure based upon Fourier Prediction

Figure 6. *Failure alerts triggered by anomaly detection*

Hardware is back

Dennard scaling is ending, and with it Moore's "Law," just at the moment when large-scale statistical modeling is becoming critical for research, government, and business. Architectural efficiency, across processors, accelerators, memory, and networking, will be the new driver of speed in computing.

by Ulrich Drepper

About the Author
Ulrich Drepper leads Red Hat's research and future vision on artificial intelligence, machine learning, and hardware innovation.

The software industry had a good run for 20 or more years: computers got faster every year while preserving backward compatibility. Innovations in both semiconductor technology and processor architecture were large enough to encourage buying a new computer every year or two.

This meant that developers could create ever more complex programs—and users could demand solving ever larger problems—without waiting longer for completion of the task. Processors gained parallelism: first in the micro-architecture, in the form of SMP, and then through multi-threading. Memory sizes increased many times over, and, as a result, so did the size of working sets that could be handled without falling back to storage devices. Life was great.

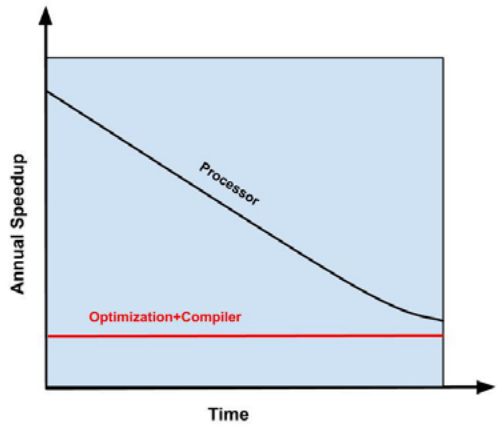


Figure 1. The flattening speedup curve

The shrinking process node sizes that drive the advances of processor speed are harder to develop after each step, and the gains are smaller. It is still possible to use less power for the same number of transistors, but the leakage is significantly higher relative to the total power needs. The node size is smaller, but some chip structures—for reasons such as power density and speed—cannot be arbitrarily shrunk. Dennard scaling is coming to the end and with it big gains in automatic performance, which means that speedups through careful program optimization and compiler optimizations are now not negligible anymore, compared to hardware speedups.

NEW PROGRAM NEEDS

As if the flattening speedup curve were not bad enough, the world at large started to pay attention to statistical modeling. The success of certain Internet businesses forced organizations to pay attention. The effective[1] techniques require enormous amounts of compute power, which opened a gap between users' requirements and what the existing hardware/software ecosystem could provide.

With the emergence of machine learning (ML)/AI/statistical learning, the number of organizations that rely on high-performance computing (HPC) dramatically increased. One could even argue that pretty much every organization is now using HPC. What once was the domain of specialized organizations and institutions (*e.g.*, university and national laboratories, the oil and gas industry) suddenly became everyone's problem.

Organizations with experience in HPC worked hard to become experts in squeezing out performance from their systems. They invested in specialized hardware (e.g., for networking), spent more on fast storage, and employed personnel to get even the hard-to-achieve speedups from optimizing their program code.

This is no practical path for the general public.

QUICK FIXES AND PROBLEMS

The computer industry has reacted to the changing environment of hardware reality and new compute demands.

One of the successful ML techniques, neural networks (NN), to a large extent relies on linear algebra. This was fortuitous because linear algebra accelerators in the form of 3D graphics cards have existed for a long time. Subsidized by gamers' need for better graphics, the technologies were adapted for compute needs and specialized software to ease writing NN on these GPGPUs (general purpose graphics processing units). Some technology firms even go beyond the graphics cards designs and focus on the needs of NNs. These technologies fulfill the compute needs that are hard to satisfy with CPUs.

All is not peachy, though. Data sets are large and have to be transported in and out. The DRAM (Dynamic Random Access Memory) of GPGPUs, because of the extreme performance requirements, is miniscule compared to what is supported on a workstation or server. This makes solving the problems tricky, because it is necessary to couple many machines with several GPGPUs or other accelerators so that they can efficiently exchange and update shared data. Solving this efficiently requires significant knowledge, energy, and time. None of which are generally available.

The only realistic substitute for all this is an inefficient solution: compensate for duplicated or inefficient work and communication by scaling horizontally. This is made possible by the rise of public clouds, which allow you to avoid buying the large number of expensive machines needed for the inefficient compute by renting them instead. If problems that require such treatment are few and have to be solved *right away,* this is likely the right path.

The only realistic substitute for all this is an inefficient solution: compensate for duplicated or inefficient work and communication by scaling horizontally.

[1] Effective because they do not require large amounts of curated data

One way forward is to use reconfigurable hardware, through rewiring bigger building blocks inside a processor to model the data flow of the problem and avoid data at rest.

But the number of ML and other high-compute projects rises. Every organization creates more and more data every day. The advent of 5G communication technologies means that endpoint devices are no longer limited by narrow channels to datacenters. This raises the expectation that one can get access to expensive analysis and results at any time and for many more problems.

The result is that renting the machines might not be possible—and in many cases, not economically viable. It is necessary to compensate for at least some of the increase in demand with increasing efficiency, and this requires drastic changes, both in hardware and, as a result, in software as well.

MORE EFFICIENT TECHNOLOGIES

Forward-looking enterprises are at the vanguard of using new technologies, and they feel the pain. Through cloud offerings, customers can at least fulfill the needs of their organizations, but this does not mean we in the software industry can stand still. We want to increase our customers' efficiency and therefore spend significant resources to improve the compute environment. The innovations on the software side have been described elsewhere. Here we will focus on changes coming on the hardware front: more fundamental changes, in more significant areas, are coming than we've seen in many years. Specifically:

- Improvements to processor technologies
- More, better accelerators
- New memory and storage technologies
- More efficient and performing networking

None of these changes can be done transparently to the software while achieving all of the potential speedups. All organizations need to pay attention.

NEW CPU TECHNOLOGIES

Even though the automatic improvements in CPU speed are slowing, that does not mean that the total compute power of a CPU is not dramatically increasing. The last years have shown significant improvements in the technology to connect dies, and CPU manufacturers are using these technologies to create CPUs with large numbers of cores and large amounts of cache memory. Without this ***chiplet technology*** such CPUs would be prohibitively expensive due to the low yield, but today there are processors with 64 or more cores available.

Chiplets will also enable functionality on the same CPU package that requires different manufacturing and/or design technologies. CPU cores alongside large-scale GPGPUs (not just small integrated graphics units) are possible, just as with integrating FPGA. This means better and faster sharing of resources, such as memory, between the different components.

Better connectivity to the outside is also needed. The next revision of the standard protocols for communication with devices has just started its way into available machines (PCIe4), and the following revision, which again doubles performance, is already specified. More I/O devices, such as network interfaces, make their way directly into the CPU package or even onto the CPU die.

The instruction set of CPUs is optimized for more specific workloads like NNs, which could redraw the line beyond which it is worth offloading compute tasks to specialized accelerators, as opposed to keeping the NN computations alongside the normal program code.

NEW, BETTER ACCELERATORS

Today, accelerators are dominated by linear algebra engines in the form of GPGPUs or more generalized chips (tensor engines). Numerous large organizations and small startups try to define the next ASIC that can accelerate more computations better for less cost, and this undoubtedly will lead to some improvements.

A problem is that much of the focus is on one type of acceleration (linear algebra), because it is easy to implement functions in hardware and program them in software. But not all problems can be solved efficiently using these technologies. Not even all NN techniques can be accelerated at the same level (see RNNs). More important still: the state of the art and the users' requirements are not static, and changing ASICs is hard.

One way forward is to use ***reconfigurable hardware***. This could be done through rewiring bigger building blocks inside a processor to model the data flow of the problem and avoid data at rest.

Another way is to start out with small, simple components and build a processor or processing engine from the ground up. This is possible with FPGAs, which have been in use to solve problems unsolvable by normal CPUs (e.g., in backbone routers) for decades.

Both of these possible technologies have in common that the same work can be performed more efficiently because only the gates needed to solve the problem are actually powered. Compare this with a general purpose CPU, which has to deal with a lot more overhead and the resulting inefficiency. The issue of efficiency is even more severe for edge devices, where power is more limited or the device may even be battery powered.

The second main advantage is the inherent parallelism of hardware in case there are no dependencies. CPUs and GPGPUs can execute as many independent tasks as there are cores/ hardware threads. Hardware like FPGAs is only limited by the total number of logical units (lookup tables [LUTs] in FPGAs), and those number in the billions these days.

Accelerators are also better connected to the host, each other, and other machines. GPGPUs that act as PCIe masters can communicate directly with the outside world without the involvement of the host. They can communicate with high rates amongst its kind within the machine, and advances in SR-IOV implementations on the accelerator and host side lead to disappearing boundaries between the memory of the host and that of the accelerators.

As related to programming, it is possible to transparently target from a program running on the host any and all accelerators, depending on availability and the specific working set. Abstraction in programming and better support in programming languages allow compilers to generate efficient code not just for the host. Runtime technologies make deployment simple even in heterogeneous compute environments with different hardware throughout the datacenter. Better tools for hardware programming make it possible for software developers to target reconfigurable hardware and experience the same development cycle.

NEW MEMORY AND STORAGE TECHNOLOGIES

The size of working sets has always been growing, but with the increased use of ML and its reliance on large amounts of data to produce good models, the problem of fitting the data onto a computer is getting more severe. The size of the working set that can fit into directly addressable memory has always been a driving factor at the top end of computers.

Even with increased capacities of an individual machine, networking will always be a major factor in a machine's performance.

DRAM technology steadily advances, with doubling of the bandwidth and maximum module capacity every few years. The access latency, however, does not decrease as impressively as that. More memory channels can lead to more concurrency, but that would have to be exploited by the program. Additionally, the fundamental access sequence does not change, which means that truly random accesses are slow because they cannot take advantage of the CPU caches.

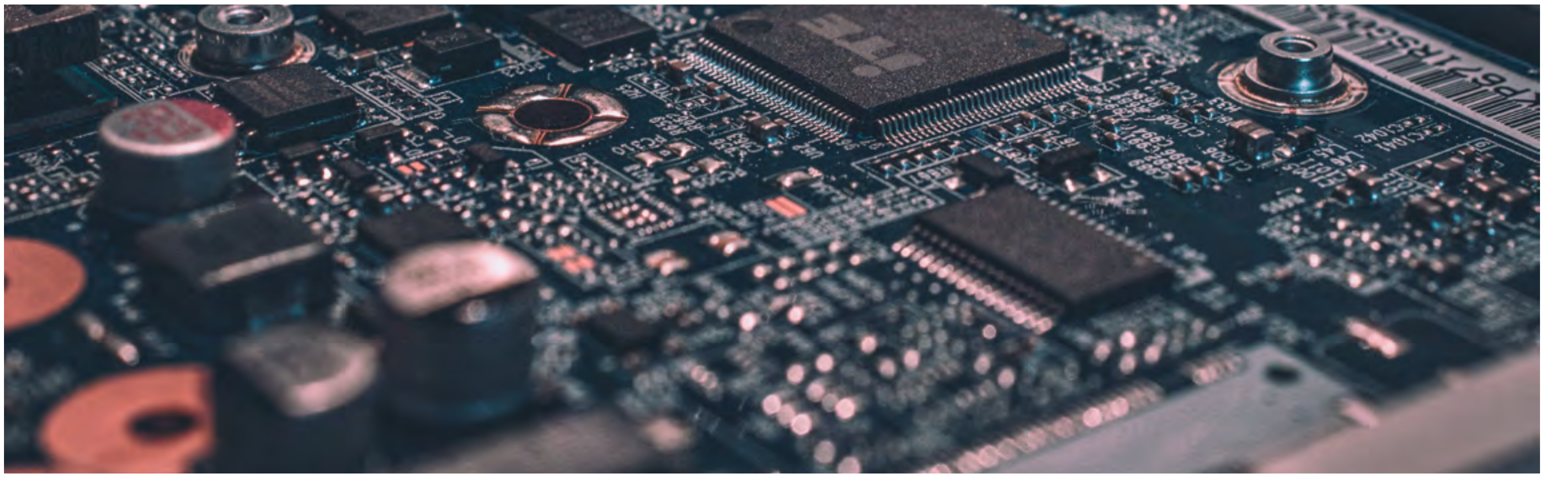
Investigations into technologies to accelerate—for instance, using tables in both row and column form at the same time—are ongoing but not yet available.

When the amount of DRAM memory that has to be available can be (severely) limited it is possible to connect the DRAM to the processor in a more efficient way: High-Bandwidth Memory (HBM) used with GPGPUs connected to the processor, using an interposer that directly connects the dies. This allows many hundreds of connections to be made from the HBM stack to the processor as opposed to the 64 data lines connecting a DDR memory module to the CPU. The protocol can also be faster because of better signal integrity, which leads to lower latency and more throughput. But HBM is only available in the package (for cost and heating reasons) in amounts that are tiny compared to the amount of memory that can be attached directly to a CPU.

Increasing the amount of DRAM attached to a CPU is also limited by power/heat reasons as well as how much memory can be reasonably supported. The next best thing is to use a different technology than DRAM that is more energy efficient. Several such technologies have been developed and deployed. The reduced energy use requires that the memory is non-volatile and requires no energy to keep its state. Given that such memory costs less energy to run, it can be many times larger than what is realistic with DRAM, and combined with the non-volatile nature this opens the opportunity to use the extended RAM as storage. In fact, most NVRAM solutions can be used as

- an extension of RAM by transparently using the DRAM as cache for the random accesses to the directly accessible NVRAM; or
- a fast storage device while continuing to use DRAM as before, but now with the possibility to swap in and out much more quickly the parts of the working set that do not fit into DRAM.

The flash NVRAM technology, which has been available for a long time now, is not suitable to implement NVRAM. It is too slow and its lifetime is too limited. Newer technologies such as Phase Change Memory (PCM), Magnetoresistive RAM (MRAM), and 3D XPoint are needed. The good news is that those technologies can then, cost allowing, be used for storage devices that



previously used flash. The resulting improvements in read and write speed are certainly welcome. In addition, the connectivity of some of the devices can be optimized. M.2 drives are not accessed through the SATA or SCSI protocol and a connected controller but are instead connected to PCIe lanes and can be directly read by the CPU, further decreasing latency and increasing throughput.

MORE EFFICIENT AND PERFORMING NETWORKING

Even with increased capacities of an individual machine, networking will always be a major factor in a machine's performance. At the very least, data has to be transferred in and out, often in a time-sensitive manner. At other times it is the throughput that is limited.

Individual network interfaces are getting faster. While today many organizations still have to make the jump from 1Gb/s interfaces to 10Gb/s, even faster versions with 50Gb/s or even 100Gb/s are on the horizon. These speeds will be necessary—especially if the number of connecting devices is exploding as expected with the adoption of 5G network technologies. If every little device has a network interface and wants to access services from a provider, the latency for individual requests will be a major factor.

Instead of just relying on efficient handling of network traffic in the traditional way (just faster), it will in many situations be necessary to break the abstraction of the network stack. The implementation of SDN controllers already bypasses the OS kernel. This is just one of the possible applications that can directly consume large amounts of network traffic and needs to handle it with low latency. Trading applications or traffic or industrial controllers are other possibilities. In many cases the work done for each request or data packet received is minimal, and the overhead of the transfer into the main memory for consumption of the OS kernel or a bypass network stack is prohibitive.

Intelligent NICs can help solve the problem. If user-defined code can be deployed onto devices connected to the network and they can receive or inspect incoming networking traffic (and send out replies), no communication outside the device has to happen. Such technology exists in the form of FPGAs that are deployed in the so-called Bump-in-the-Wire (BitW) configuration. With the addition of support for multi-tenancy on the FPGA, it is possible to deploy them in datacenters for general use.

CONCLUSION

The hardware environment is stagnating in some areas (per-core CPU performance), thriving in others (5G networking), and it receives different requirements from demanding applications (ML). The result is that hardware to be used in datacenters is today changing more and in different directions than in the last several years. This is at once exciting and challenging, requiring research and development from Red Hat and from its partners in both hardware and software.

Research project updates

Faculty, Ph.D. students, and U.S. Red Hat associates in Israel are collaborating actively on the following research projects. This quarter we highlight collaborative projects at Technion–Israel Institute of Technology, Tel Aviv University, and the Interdisciplinary Center Herzliya. We will highlight research collaborations from other parts of the world in future editions of the Research Quarterly. Contact academic@redhat.com for more information on any project described here.

PROJECT: CEP (Complex Event Processing)

ACADEMIC INVESTIGATORS:
Prof. Assaf Schuster (Technion)

RED HAT INVESTIGATORS:
Ilya Kolchinsky

Researchers plan to build a scalable, real-time cloud-based CEP engine capable of efficiently detecting arbitrarily complex patterns in high-volume data streams. The engine will be implemented on top of Red Hat® OpenShift® Container Platform and will be applicable to any domain where event-based streaming data is present. One goal of the project is creating an open source project/community based on the above engine. The researchers also hope to advance the state of the art in the area of complex event processing and combine academic research with the implementation and deployment of novel CEP mechanisms and techniques in the above framework.

For more information, see Ilya Kolchinsky's article, "Multi-pattern detection over streaming data" in *Red Hat Research Quarterly* 1:4 (research.redhat.com).

PROJECT: Predictive Analysis-Fault Tolerance

ACADEMIC INVESTIGATORS:
Massachusetts Open Cloud

RED HAT INVESTIGATORS:
Gagan Kumar (Boston)
Ilana Polonsky (Tel Aviv)
Parul Singh (Boston)
Shirly Radco (Tel Aviv)
Steven Rosenberg (Tel Aviv)

The goal of this project is to build an algorithm that would utilize predictive analysis technology to create a state-of-the-art fault-tolerance system that can lead toward the ability to "predict," based upon past events, if and when faults such as component failures may occur. The target platform is the Mass Open Cloud (massopen.cloud), a public-cloud alternative for the academic research community. (Joint work with Boston)

PROJECT: Electroencephalography (EEG) Feature Extraction

ACADEMIC INVESTIGATORS:
Lubov Blumkin, M.D. (Sackler School of Medicine, Tel Aviv University)

RED HAT INVESTIGATORS:
Boris Odnopozov

This research is meant to enable improvement of the management of patients with ESES.

Electrical status epilepticus during slow wave sleep (ESES) is a rare age-related disorder that appears in childhood, usually between ages 4 and 9 years, and disappears by puberty. The disorder is characterized by a combination of multiple types of seizures and continuous spike wave discharges during NREM sleep on an EEG. The ESES syndrome may present with frequent multiple type seizures, although subclinical seizures may occur.

The treatment is aimed at controlling the seizures and the epileptic activity in order to prevent sequelae. Inappropriate treatment may lead to persistent neuropsychological sequela (language capacity, intellectual level, memory, behavior) as well as motor impairment.

The ESES disorder is usually resistant to multiple antiepileptic drugs. Recurrent sleep EEG study during therapeutic trials is needed for detecting the efficacy of medications. Currently this requires monitoring the child during sleep at a sleep laboratory or EEG institute using a full set of electrodes, which is both an expensive and a burdensome procedure. As a result, physicians do not have feedback on the efficacy of the medications, which makes the management ineffective.

Using Open Data Hub (opendatahub.io) as the AI platform, we attempt to see if we can detect ESES using only frontal electrodes. This can allow much easier and more effective home-based monitoring that will allow practitioners to better administer the drugs and make the treatment more effective.

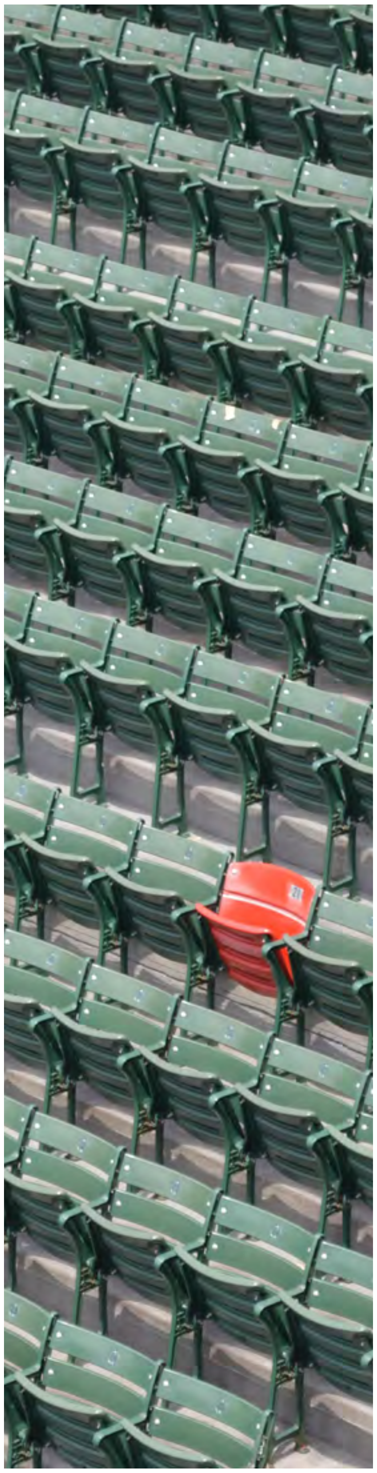
PROJECT: Ceph: Wire-Level Compression-Efficient Object Storage Daemon Communication for the Cloud

ACADEMIC INVESTIGATORS:
Prof. Anat Bremler-Barr
Maya Gilad
(The Interdisciplinary Center Herzliya)

RED HAT INVESTIGATORS:
Josh Salomon

The project's purpose is to reduce storage network traffic (object, block, etc.) for the following cases: between the failure domains in cost-sensitive environments such as public clouds, and between nodes in cases where the network bandwidth is the bottleneck of the node performance. We have divided the project into 3 milestones: applying compression for data transfer between different datacenters, enabling/disabling compression given hints from the client, and minimizing compression efforts when data is non-compressible.

Find out more at research.redhat.com. RH RQ



intel AI
AI ON INTEL®
intel XEON® PLATINUM inside™
NOW BUILD THE AI YOU WANT ON THE CPU YOU KNOW.
Learn more at ai.intel.com