RH RQ

Bringing great research ideas into open source communities

Anat Bremler-Barr

when one plus one makes more than two

Translation layers for the cloud

Planting research seeds

+

Demystifying scheduling latency

New mentorship program: Irit Goihman and Liora Milbaum realize potential when experience meets passion

Red Hat
Research Quarterly
Volume 3:1 | May 2021 | ISSN 2691-5278

THE UNIVERSAL AI SYSTEM FOR HIGHER EDUCATION AND RESEARCH

NVIDIA DGX A100

Higher education and research institutions are the pioneers of innovation, entrusted to train future academics, faculty, and researchers on emerging technologies like AI, data analytics, scientific simulation, and visualization. These technologies require powerful compute infrastructure, enabling the fastest time to scientific exploration and insights. NVIDIA® DGX™ A100 unifies all workloads with top performance, simplifies infrastructure deployment, delivers cost savings, and equips the next generation with a powerful, state-of-the art GPU infrastructure.

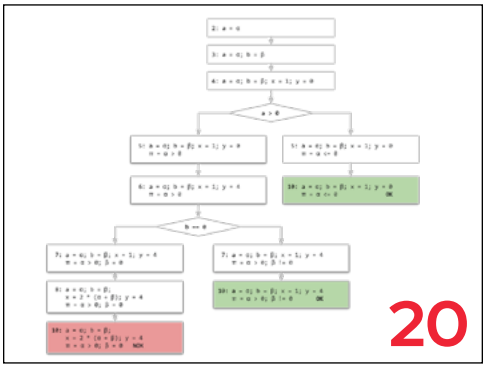
Learn More About **DGX** @ nvda.ws/dgx-pod
Learn More About **DGX on OpenShift** @ nvda.ws/dgx-openshift

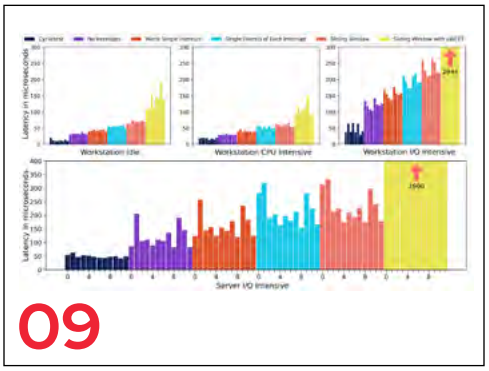
RESEARCH QUARTERLY

VOLUME 3:1

Red Hat

Table of Contents





Departments

Features

ABOUT RED HAT Red Hat is the world's leading provider of open source software solutions, using a community-powered approach to provide reliable and high-performing cloud, Linux, middleware, storage, and virtualization technologies. Red Hat also offers award-winning support, training, and consulting services. As a connective hub in a global network of enterprises, partners, and open source communities, Red Hat helps create relevant, innovative technologies that liberate resources for growth and prepare customers for the future of IT.

NORTH AMERICA
1 888 REDHAT1

EUROPE, MIDDLE EAST, AND AFRICA
00800 7334 2835
europe@redhat.com

ASIA PACIFIC
+65 6490 4200
apac@redhat.com

LATIN AMERICA
+54 11 4329 7300
info-latam@redhat.com

facebook.com/redhatinc
@redhatnews
linkedin.com/company/red-hat

From the Director

Expanding the impact of open source

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.

As we begin our third year of *RHRQ* I am in a celebratory mood—unusual, for 2021, but I think appropriate. I've just finished rereading our interview with Professor Anat Bremler-Barr of the Interdisciplinary Center in Herzliya, Israel. What really struck me about the interview is that she has independently validated the model we have been championing since the beginning of Red Hat Research, and perhaps even the beginning of Red Hat: If you can work to prove your thesis upstream in the open, your work will be stronger and more relevant to what is happening in software. I don't think it's likely we will change the world of research with this model, but I do think the approach is opening up researchers to the idea of sharing ideas sooner and more broadly.

But what kind of ideas should we be sharing? Open source is focused on sharing source code so that many people can participate in improving it—many eyes are (usually) better than just two. However, sometimes looking at code is not sufficient. Two of our technical features this issue focus on formal analysis of software, which is generally held to be impossible or at the very least impractical for real-world programs. Yet in both of these cases we find that using formal techniques to solve very specific, bounded problems can be quite useful. Daniel Bristot de Oliveira finishes his three-part series on proving the real-time-ness of the Linux kernel using finite automata, an ingenious application of formal techniques that shows their practicality when applied the right way. Heinrich Lauko's very technical piece shows how to leverage a general analysis framework developed by researchers in Brno to solve a very specific fuzzing problem. When tools like these become commonplace in the open, our conversations go from line-by-line analysis of code to broader discussion of its quality, which is useful for all concerned.

At Red Hat Research we spend a lot of time focused on our own experimental cloud, the Mass Open Cloud. One of the great things about having a cloud to play with is that it allows for research to answer scale questions that simply can't be answered using a laptop. Peter Desnoyer's exposition of his team's work on Ceph storage architecture is a nice example of how useful this is. The team has been able to quickly test concepts on the Mass Open Cloud that are going to turn out to be really beneficial for the Ceph project and cloud storage in general.

Finally, I'm really pleased to be able to say that we're dramatically expanding our flagship research partnership with Boston University (see the full press release online, bit.ly/BURedHatOpenHybridCloud). I am privileged to be working with such a productive and adventurous group of researchers at Boston University, and grateful for the contributions Red Hat engineers are making to make the partnership great. I look forward to reporting important research results from our partnership here in the months and years to come.

Red Hat

Telemetry Working Group looks at observability

by Gordon Haff

A new working group is tackling observability in production.

Observability has become an increasingly hot topic given the challenges of reliably operating distributed systems such as those in Kubernetes environments. The term can cover a lot of ground, but a typical definition of observability spans metrics, tracing, and logging. Even if monitoring is often considered as something distinct, it's at least closely related. A key part of observability is the automatic collection and transmission of data. In other words, telemetry.

There is no shortage of open source projects in this space. However, the production-level testing and refinement of these tools—together with their associated procedures and datasets—has been much less common in an integrated multi-tenant open environment. That's the problem that the new Telemetry Working Group (WG) is tackling.

A variety of other initiatives are related to the Telemetry WG. OpenInfra Labs (openinfralabs.org, under the Open Infrastructure Foundation) is hosting the working group. Operate First (operate-first.cloud) will house the experiments and research associated with the group. Initially, the group will focus on Kubernetes, but their work may be extended to other high-performance computing environments over time. The Mass Open Cloud (MOC; massopen.cloud), which sponsors and hosts a large portion of Operate First, is also involved, as is the New England Research Cloud (nerc.mghpcc.org).

> It's a cross-research university, cross-company, and cross-open source effort.

It's a cross-research university, cross-company, and cross-open source effort. This specific initiative was first kicked off by Boston University's Michael Daitzman, although there have been other discussions and work going on in this general area for a while. It's now co-chaired by Tufts University's Raja Sambasivan and Marcel Hild, a manager of software engineering in Red Hat's Office of the CTO.

The group's goals are as follows:

- Create open datasets for research
- Provide access to a platform for telemetry research
- Define and implement a standardized application stack, i.e., the gold standard
- Define research problem statements around telemetry

continued on pg. 7

News

Big data, security certification, and FPGAs: 2021 Red Hat Research Days have begun

by Gordon Haff

This year has already brought us several Research Days discussions streaming around the world. They have covered topics as diverse as big data stream processing, analyzing security certification reports for potential device and product vulnerabilities, and using open source tools to program FPGA applications.

Ilya Kolchinsky, Senior Software Engineer at Red Hat in Israel, kicked things off on March 2 by describing a growing problem. A large number of data-driven systems and applications have become an integral part of our daily lives, and this trend is accelerating dramatically. An estimated 1.7 MB of data is created every second for every person on Earth, for a total of over 2.5 quintillion bytes of new data every day, projected to reach 163 zettabytes by 2025. In addition to the growing volume, velocity, and variety of continuously generated data, novel technological trends such as edge processing, IoT, 5G, and federated AI bring new requirements for faster processing and deeper, more computationally heavy data analysis. Hence the challenge: old-school data processing mechanisms are no longer enough.

In a spirited discussion with conversation leader Oren Oichman, Senior Cloud Consultant at Red Hat, Ilya explored potential ways to analyze this data dynamically, with an approach called Big Data Stream Processing (BDSP). BDSP uses a variety of methods for scalable and efficient data processing that do not rely on traditional databases for storing and processing the data. Ilya and Oren discussed specific examples of real-life applications that can greatly benefit from incorporating BDSP capabilities. In particular, he covered on-the-fly detection of complex patterns in streaming and stream-oriented machine learning and data mining.

> The research group hypothesized that a solution to P^4 could thus be built using existing open source tools...

Later in March, Petr Švenda, Faculty of Informatics, Masaryk University in Brno, Czech Republic, noted that long security certification reports can be a trove of publicly available data about proprietary devices and other products otherwise available only under NDA. While downloading and reading a single certificate is easy, reasoning about the characteristics of the whole associated ecosystem, which might have more than ten thousand certified devices, is much harder. Petr's talk addressed using an open source tool for automatic analysis of publicly available certification reports to answer questions like these: Are there observable systematic differences between the Common Criteria

and FIPS 140-2 certificates? Can I quickly find out whether my device is using a certified component recently found vulnerable? Most importantly, can we measure and quantify the extent to which the whole process is actually increasing the security of products being certificated?

Finally, Martin Herbordt, Professor of Electrical and Computer Engineering at Boston University, and Robert P. Munafo, a PhD candidate there, discussed practical plans for programming FPGAs (Field Programmable Gate Arrays) in the datacenter. FPGAs—flexible chips that can be "programmed" again and again with different code paths—are now essential components in the datacenter and on the edge, with millions currently deployed. FPGAs are in a wide variety of system components and provide such critical functions as SDN, encryption/decryption, and compression. Yet for nearly all system providers, much less system users, programming these FPGAs is impossible. Martin and Robert, along with Red Hat Senior Data Scientist Ahmed Sanaullah, who also joined the conversation, have been working to enable high-level language programming for FPGA application development, especially in the datacenter and at the edge, exclusively using existing open source tools.

Previous research by Martin and others showed that current compilers could deliver excellent FPGA performance for arbitrary C code, but that this capability was brittle, inconsistent, and required special programmer expertise to extract. Taking advantage of the flexibility and performance potential of FPGAs has typically required either expensive specialized engineering talent, commercial proprietary C-to-hardware tools that yielded demonstrably poor performance, or both. This is the performance portability programmability problem (P^4).

P^4 can be reduced to the problem of generating the correct sequence of optimizations for a particular input code and target architecture. The research group hypothesized that a solution to P^4 could thus be built using existing open source tools, primarily based on the GNU C Compiler (GCC). In particular, they discussed an ongoing project that aims to use machine learning to control a newly customizable version of the GCC to automatically determine optimization pass ordering for FPGA targets specifically, and thereby improve performance as compared to existing proprietary C-to-FPGA methods. This research is continuing as part of the Red Hat Collaboratory at Boston University (bu.edu/rhcollab).

Telemetry Working Group
continued from pg. 5

- Iterate over implementations of solutions on those problem statements

Another explicit goal is to *not* create new open source projects. As Hild puts it, "We have a large number of projects solving similar enough problems. The challenge these days lies in connecting these projects and operating these projects in a real environment." He adds, "We don't want to do everything in a lab; that's a controlled environment. And controlled environments are only so good."

A core premise of the working group from the beginning has been to operate in public and to make any code open source over time, even if it's not at the very beginning, as well as any data that does not include personally identifiable information. Anyone is welcome to participate. Meetings are recorded and can be accessed via the Telemetry Working Group Playlist on the MOC YouTube page (bit.ly/telemetryWG). The group's repository is on GitHub (github.com/open-infrastructure-labs/telemetrywg).

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.

Boston University and Red Hat partner to support the New England Research Cloud

Boston University is collaborating with other area universities to extend the success of the Mass Open Cloud (massopen.cloud) in supporting critical research projects to a public cloud that will serve research needs throughout New England in the United States. The New England Research Cloud (NERC; nerc.mghpcc.org) aims to deliver production-quality cloud resources and services to its research communities throughout the region. NERC will be hosted in the Massachusetts Green High Performance Computing Center (MGHPCC; mghpcc.org).

NERC will serve a dual purpose. First, it will host research projects requiring hyperscale computing resources. Second, it will serve as a laboratory of sorts, where every aspect of the operations of such a complex cloud can be studied and lead to further development of AIOps. As a regional center of excellence in research into clouds, NERC will provide a stable and accessible space for open source communities to develop operational knowledge around cloud infrastructure in partnership with the Operate First (openinfralabs.org) initiative, hosted by the Open Infrastructure Foundation.

Red Hat is donating over $500 million in software subscriptions to Boston University, which will form the foundational operating stack of the NERC infrastructure. With this contribution, Red Hat endeavors to speed breakthroughs in cloud-based technologies and related open source projects, while building critical skills needed in the next wave of IT professionals.

NERC is part of other projects grouped under the umbrella of the Open Cloud Initiative (OCI), which includes the Mass Open Cloud (MOC), Northeast Storage Exchange (NESE; nese.mghpcc.org), Open Cloud Testbed (OCT; massopen.cloud/connected-initiatives/open-cloud-testbed), and Open Storage Network (OSN; openstoragenetwork.org).

Already, these projects have had meaningful impacts, including:

- Significant contributions to open source storage (bit.ly/opensourcestorage), operating systems, and security projects
- The development of critical advancements such as the ChRIS Research Integration Service (bit.ly/chRISplatform) in collaboration with Boston Children's Hospital. ChRIS is a web-based medical image platform developed using Red Hat technologies on the MOC that provides a distributed user interface that is designed to enable real-time collaboration between clinicians and radiologists around the world
- Millions of dollars in research funding, including a recent grant from the National Science Foundation (NSF) Division of Computer and Network Systems (bit.ly/NSFtestbed) to help fund the development of the OCT, a national cloud testbed for research and development of new cloud computing platforms
- Efforts to close the education skills gap so that students and graduates have the ability to work with premium, industry-standard, open source software

RESEARCH
QUARTERLY

VOLUME 3:1

Red Hat

Feature

Demystifying real-time Linux scheduling latency

by Daniel Bristot de Oliveira, PhD

This is the third of a series of three articles about the formal analysis and verification of the real-time Linux® kernel. Read the first article in RHRQ 2:3 *and the second article in* RHRQ 2:4.

Scheduling latency is the principal metric of the real-time variant of Linux, and it is measured using the `cyclictest` tool. Despite its practical approach and contributions to the current state of the art of real-time Linux, `cyclictest` has some known limitations. The main constraint arises from the opaque nature of the latency value provided by `cyclictest`. The tool only provides information about the latency value, without providing insights on its root causes. This fact, along with the absence of a theoretically sound description of the in-kernel behavior, raises some doubts about whether Linux can really support the "real-time" description.

A common approach in real-time systems theory is categorizing a system as a set of independent variables and equations that describe its integrated timing behavior. The first article of this series (see *RHRQ* 2:3) presented the *thread synchronization model,* which is composed of a set of formal specifications that define the behavior of the system. This article leverages the specifications of that model to discover a safe bound for the scheduling latency of Linux. It also uses findings from the second article of this series (see *RHRQ* 2:4), in which an efficient verification tool was developed, by using the same method to capture the values for variables that compose the scheduling latency on a real system and identify the root cause of high latency values.

FROM INFORMAL TO FORMAL

The latency experienced by a *thread instance* is, informally, defined as the maximum time elapsed between the instant in which it becomes ready while having the highest priority among all ready threads and the instant in which it is allowed to execute its own code after the context switch has already been performed.

A common approach in real-time systems theory is categorizing a system as a set of independent variables and equations that describe its integrated timing behavior. The first step in this approach is to define the task model of the system. In this work, the task model is composed of three levels of tasks: the NMI, the IRQs, and the threads. The system has a single NMI, a set $IRQ = \{IRQ_1, IRQ_2, ...\}$ of maskable interrupts, and a set of threads $\tau = \{ \tau_1, \tau_2, ... \}$. The NMI, IRQs, and threads are subject to a scheduling hierarchy in which NMI always has a higher priority than IRQs, and IRQs always have higher priority than threads.

Given a thread τ_i at a given point in time, the set of threads with a higher priority than τ_i is denoted by $HP(\tau_i)$. Similarly, the set of tasks with priority lower than τ_i is denoted by $LP(\tau_i)$. From the τ_i thread perspective, all IRQs and the NMI belong to $HP(\tau_i)$. Although the schedulers might have

About the Author
Daniel Bristot de Oliveira is a principal software engineer at Red Hat working in the development of real-time features of the Linux kernel. Daniel has a joint PhD degree in Automation and Systems Engineering at UFSC (BRA) and in Embedded Systems at the Scuola Superiore Sant'Anna (ITA). He is also a post-PhD researcher in the Retis Lab at the Scuola Superiore Sant'Anna.

threads with the same priority in their queues, only one among them will be selected to have its context loaded and, consequently, start running. Hence, when scheduling, the schedulers elect a single thread as the highest priority one, with all other active threads belonging to LP(τ_i).

The tasks can also influence one another via synchronization primitives. For example, a thread can postpone the execution of an IRQ by temporarily masking interrupts, or it can defer the execution of another thread in HP(τ_i) by temporarily disabling the preemption. Moreover, the scheduling operation itself influences the thread execution timeline because it is not an atomic operation.

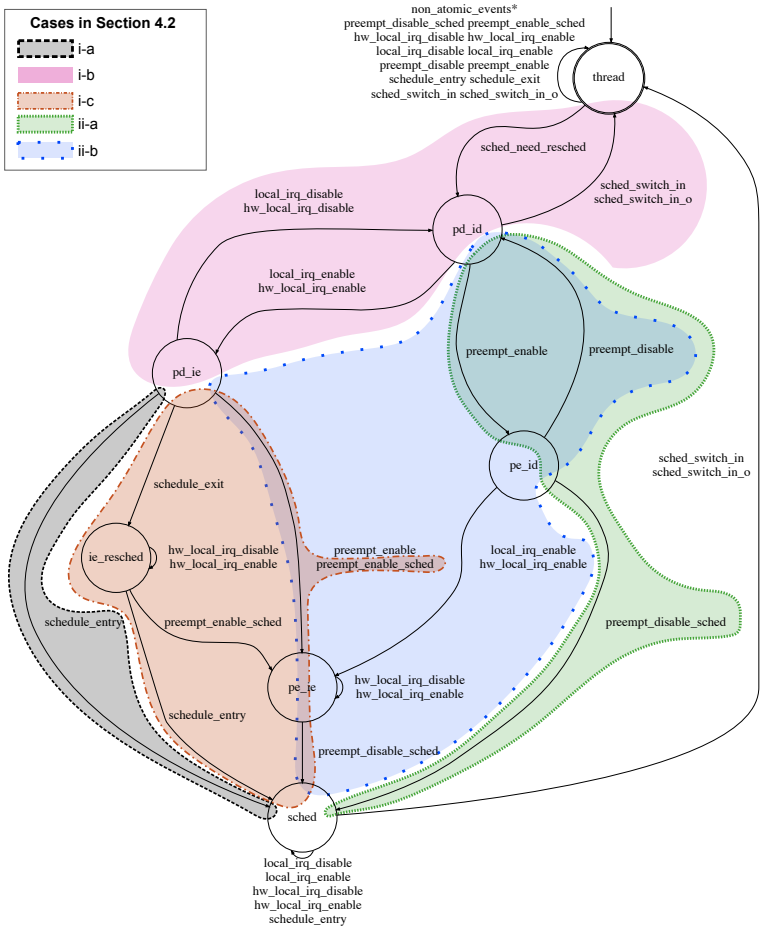


Figure 1. *Setting* `need_resched` *always causes a context switch specification.*

The complexity imposed by the different levels of tasks, the synchronization primitives, and the overhead involved in Linux threads' scheduling makes informal language inadequate for this analysis. A formal thread synchronization model is required instead. The model, which enables the composition of specifications using a deterministic format, removes the ambiguity of natural language while enabling reasoning about the system in a more analytical manner.

In this work, we have translated the specifications of the thread synchronization model into a set of properties. We then leverage these properties in an analysis that derives a theoretically sound bound for scheduling latency.

A THEORETICALLY SOUND BOUND FOR SCHEDULING LATENCY

The scheduling latency experienced by an arbitrary thread τ_i in τ is the longest time elapsed between the time *A*, in which any job of τ_i becomes ready and has the highest priority, and the time *F*, in which the scheduler returns and allows τ_i to execute its code, in any possible schedule in which τ_i is not preempted by any other thread in the interval [*A*, *F*]. We begin determining which types of entities may prolong the latency of τ_i.

In real-time theory, any time a task in LP(τ_i) delays τ_i, τ_i is said to be blocked. When the task delaying τ_i is in HP(τ_i), τ_i is said to be suffering interference. These two forms of delay were analyzed separately, starting with the blocking. Each of these forms of delay was characterized by variables and equations whose definition supports the model. The main specification used in the definition of the latency bound is shown in **Figure 1**.

Blocking time
Blocking time (referred to as *interference-free latency* in the original paper) is characterized with the following equation[1]:

$$L^{IF} \leq \max(D_{ST}, D_{POID}) + D_{PAIE} + D_{PSD}$$

Where:

- *L* means the scheduling latency
- *IF* means interference free
- *D* means the delay of
  - *ST*: the sched tail delay, which is the delay from the IRQs being disabled to cause the context switch, and the return from the scheduler
  - *POID*: the longest preemption or IRQ disabled to postpone the scheduler
  - *PAIE*: the longest time in which the preemption and IRQs are transiently enabled in the return of the preemption or IRQ enable, that will cause the scheduler execution to preempt the current thread
  - *PSD*: the longest time in which the preemption is disabled to execute **__schedule()** function.

It is worth noting that a blocking delay is only caused by other threads, which by definition belong to the sets of lower priority tasks $LP(\tau_i)$. The delay caused by IRQs and NMI is all accounted for as interference.

[1]*The development of the equations are not presented here because of space constraints. Please refer to the original paper for further information: "Demystifying the real-tIme Linux scheduling latency," presented in the 32nd Euromicro Conference on Real-Time Systems (ECRTS 2020).*

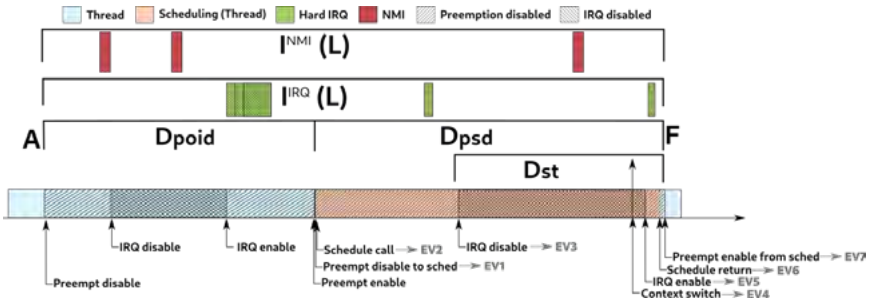
Interference
Because there is no single way to characterize these workloads, defining the interference caused by the NMI and IRQs presents a challenge. So instead of defining a single best way to compute interference from interrupts, the interference from IRQs and NMI was defined as two functions in the theorem that define the latency. The function can then be selected according to the more accurate representation of the system.

The latency bound
The scheduling latency is then defined as the sum of blocking time and interference time, as in the following equation. The *L* in both sides of the equation is resolved by solving the equation until it converges on both sides.

$$L = \max(D_{ST}, D_{POID}) + D_{PAIE} + D_{PSD} + I^{NMI}(L) + I^{IRQ}(L)$$

Figure 2 shows these variables in a timeline format, helping to illustrate the composition of the latency from *A* to *F*.

Figure 2. *Reference timeline*



RTSL: A LATENCY MEASUREMENT TOOL

As shown in the first article, it is possible to observe the thread synchronization model's events using Linux's tracing features. The

obstacle is that the simple capture of these events using trace causes a non-negligible overhead in the system, both in CPU and memory bandwidth, representing a challenge for measuring variables in the microseconds scale. However, as demonstrated in the second article, it is possible to process these events in-kernel, reducing overhead. In this article, an efficient verification method was leveraged to develop the real-time latency measurement toolkit named `rtsl`. The toolkit architecture is presented in **Figure 3**.

The *latency parser* is a kernel module that uses the thread synchronization model's kernel tracepoints to observe their occurrence from inside the kernel. The latency parser registers a callback function to the kernel tracepoints. The callback functions then pre-process the events, transforming them into one of the variables presented in Figure 2, only exporting them to the trace buffer when necessary. This reduces the overhead enough to enable the usage of the tool for measurements. For example, the record of these values adds only around two microseconds to the `cyclictest` measurements when compared to the same system running without trace.

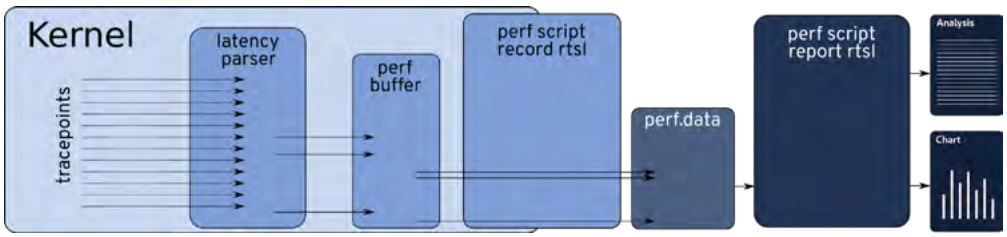


Figure 3. *Overview of the `rtsl` toolkit*

```
Interference Free Latency:
    paie is lower than 1 us -> neglectable
    latency = max(poid,    dst) + paie +   psd
            42212 = max(22510, 19312) +    0 + 19702
Cyclictest:
    Latency =     27000 with Cyclictest
No Interrupts:
    Latency =     42212 with No Interrupts
Sporadic:
    INT:     oWCET           oMIAT
    NMI:        0               0
     33:    16914          257130
     35:    12913            1843 <- oWCET > oMIAT
    236:    20728            1558 <- oWCET > oMIAT
    246:     3299         1910321
           Did not converge.
```

```
continuing...
Sliding window:
    Window: 42212
            NMI:        0
             33:    16914
             35:    14588
            236:    20728
            246:     3299
    Window: 97741
            236:    21029 <- new!
    Window: 98042
    Converged!
    Latency =     98042 with Sliding Window
```

Figure 4. *`perf rtsl` output: excerpt from the textual output (time in nanoseconds)*

The processing of the data is done in user space, using `perf` as the interface. Perf is used to capture, store, and process the data in user space. The processing phase, named `report`, produces both graphical and textual output. The textual output shows the value for each of the variables that compose the latency and the hypothetical latency for each given interrupt function. An example of the output is shown in **Figure 4**.

The textual output serves to identify how much each variable contributes to the scheduling latency. This evidence can be used then to trace the specific variable, in such a way to identify the root cause of bad values. An example of this procedure is shown in the original paper.

EXPERIMENTS

This section presents some latency measurements, comparing the results

found by `cyclictest` and `perf rtsl` while running concurrently in the same system. The experiments were executed on two systems: a workstation and a server. The *Phoronix test suite benchmark* was used as a background workload to exercise different parts of the system. One sample of the results of the experiments is shown in **Figure 5**. The workload of each experiment is explained in the legend. The colored columns represent the different metrics. The first is `cyclictest`, the second is the interference-free latency, and the next four are the hypothetical latency based on the given interrupt interference function from `rtsl`. Consistently, the proposed approach found sound scheduling latency values higher than `cyclictest` could find in the same time frame. Considering interference curves such as the sliding window, the latency values are still in the microseconds scale, even on non-tuned general purpose hardware. When considering the highly pessimistic sliding window with oWCET (observed worst-case execution time) interference, the latency is bound to the single digit milliseconds, enough to justify Linux on a vast set of safety-critical use cases.

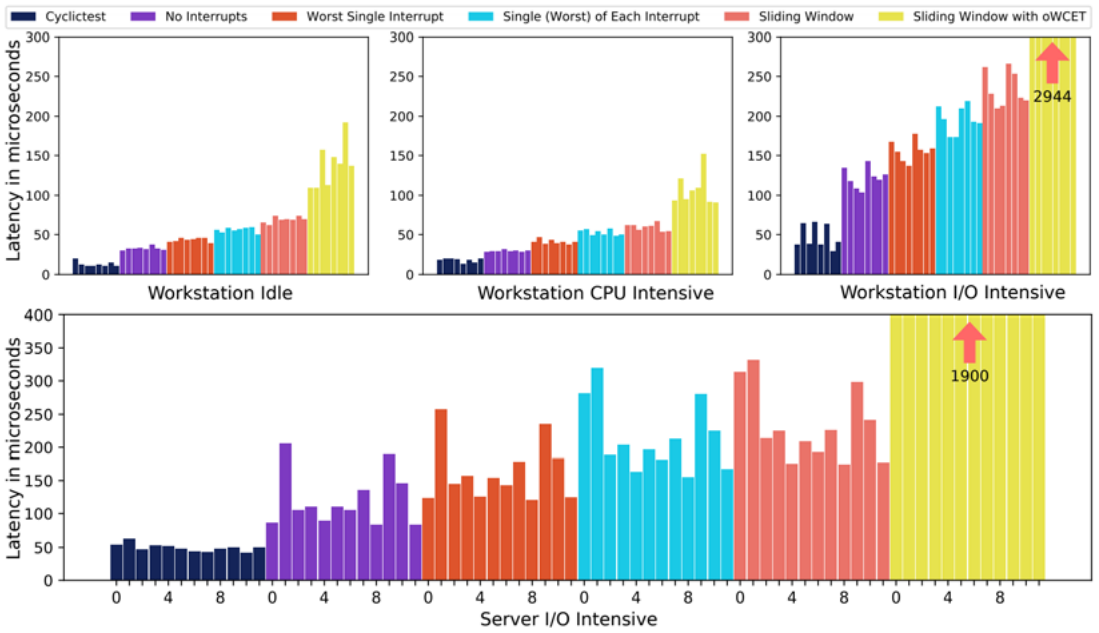


Figure 5. *`cyclictest` versus `rtsl` on multi-core environments*

FINAL REMARKS

Usage of real-time Linux in safety-critical environments, such as in the automotive and factory automation field, requires a set of more sophisticated analyses of both the logical and timing behavior of Linux. In this series of articles, we presented a viable approach for the formal modeling, verification, and analysis of the real-time preemption mode of Linux. The definition of the latency bound was the primary goal of this research. However, the complexity of Linux required the support of a formal language to abstract the complexity of the code and the development of an efficient way to monitor the relevant events. With the bases set, the mathematical reasoning about the kernel behavior was evident, resulting in an analysis accepted by the academic community, who raised this limitation of Linux more than a decade ago.

In addition to the evident results, such as the efficient runtime verification and the mathematical demonstration of the bound of the scheduling latency, this research is motivating the development of new methods for Linux analysis. Examples of ongoing research range from the usage of temporal language for runtime verification to the application of probabilistic methods for the definition of values for the variables that compose the timeline of tasks on Linux. More details about the "Demystifying the real-time Linux scheduling latency" paper are available at bristot.me/demystifying-the-real-time-linux-latency/.

Interview

When one plus one makes more than two: how open source builds a bridge between universities and industry

by Idan Levi

*Research Director and RIG leader for Israel Idan Levi speaks with **Anat Bremler-Barr**, Professor in the School of Computer Science and Vice Dean of the Efi Arazi School of Computer Science at the Interdisciplinary Center, Herzliya, Israel (IDC). Before joining the faculty of IDC, Bremler-Barr co-founded a company to provide systems that protect against Denial of Service attacks. It was acquired by Cisco systems in 2004. Her research interests are in computer networks and distributed computing, with an emphasis on security.*

About the Author

Idan Levi is the Research Director for Red Hat Research in Israel.

Idan Levi: What is the role of the IDC—and your role in particular—in supporting and expanding the open source community in Israel?

Anat Bremler-Barr: I will answer from my academic and educational perspective: I want to expose my students to the open source community and encourage them to be part of it. As open source becomes the new industry standard, it is expected by software companies that developers will have the background and skills required to utilize the potential of that field. I want the students to have up-to-date knowledge in today's open source tools as part of their education.

Open source is all about the collaborative method of software development. Students, in their studies here, rarely work in groups. However, in industry, software development is a group effort. Hence, while experiencing open source development, they get a chance to learn the skills to develop in collaboration. While working on an open source project, students build a work portfolio that is public in GitHub, which can help them when they begin to interview for jobs.

Idan Levi: What do you think industry partners and universities could do to help the open source community grow?

Anat Bremler-Barr: I think every graduate of a computer science program should have experience with open source as part of the curriculum and be part of an open source project. Philosophically, I feel that the open source ethos is very close to the ethos of academia: it's the collaborative effort of the community that makes progress for human knowledge and the general good. This openness is done without compromising the possibility of making money from work and effort. In open source, for example, businesses can succeed by using the service model, and academia is responsible for many patents and start-ups.

Idan Levi: How do you see open source being part of the curriculum? The first thing that comes

to my mind is UC Berkeley, inventing DNS and FreeBSD and really laying the groundwork for all of the open source collaboration around technology. How can universities do this today?

Anat Bremler-Barr: I think it should be integrated in mandatory courses. In operating systems, you should understand the internal elements of Linux®—see how they work and how it was built. In networking you should speak about, say, Kubernetes, OpenShift, things like that. It should be part of the curriculum.

The main problem is that open source projects are usually so big that it's hard for a student to contribute and understand the whole project. I think it can be done if we invest in it and help students understand the process, the environment, and the general framework, and then help them contribute in something small.

Idan Levi: What about universities choosing something of their own to develop? For example, SPARK started at the University of Southampton. Is that something we can do in Israel?

Anat Bremler-Barr: I'm sure. We have very experienced students in Israel because some of them begin studying after they've been in the army or worked in industry. I think at the IDC in particular we include a pragmatic approach along with the theoretical. We have a spirit of innovation, and we are more flexible and open to collaboration. It's important to have all the flavors of

"It's important to have all the flavors of computer science: the pragmatic, the theoretical, the innovative, and anything in between." —Anat Bremler-Barr

computer science: the pragmatic, the theoretical, the innovative, and anything in between.

Idan Levi: Has that belief affected how you choose your PhD students? Will you do something different from more traditional, research-oriented universities?

Anat Bremler-Barr: I don't think there is a difference there. The PhD students are all over the spectrum. You need to have theoretical faculty and students, because some of them will make the biggest breakthroughs. But you also need the more pragmatic faculty and students.

From my experience, the theoretical foundation is very important even if you wish to pursue an industry career. The theoretical foundation of computer science shapes the way you approach problems. It gives the ability to think abstractly and concentrate on the essence of the problems. Nonetheless, I think that you also need to expose the students to up-to-date practical tools and paradigms. That is why IDC now provides a dual track in Computer Science and Entrepreneurship.

In many cases it can be hard to collaborate with companies, but when you have a framework to do it, you get one plus one adding up to more than two.

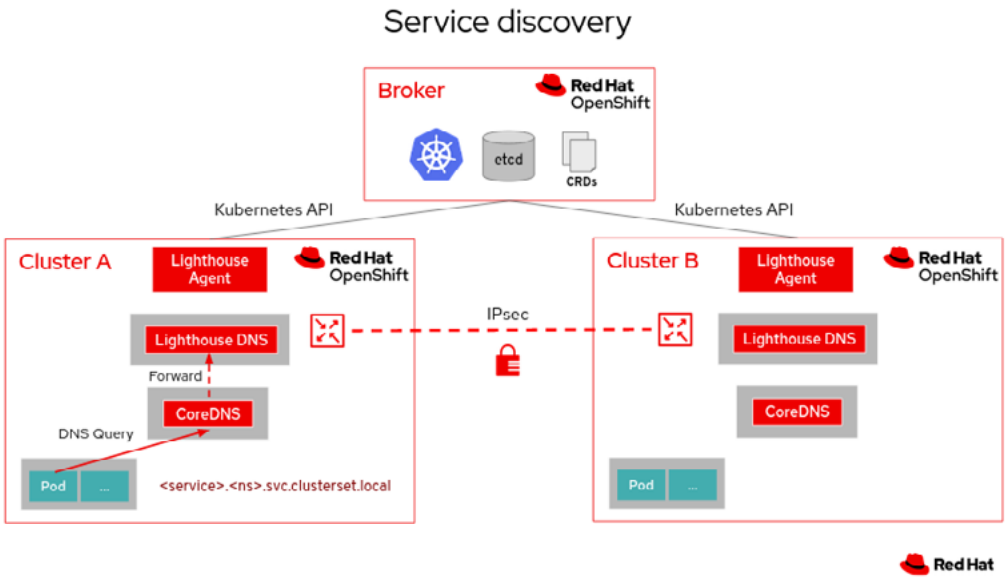
Idan Levi: What about industry? What has it been like to engage with different companies to support open source?

Anat Bremler-Barr: Right now in Israel the government is investing in helping academia and industry work together. In many cases it can be hard to collaborate with companies, but when you have a framework to do it, you get one plus one adding up to more than two. I do organize a workshop, "Project with the Industry," with companies that are part of our industrial affiliation program (IAP), in which Red Hat participates.

There is a lot of power in academia, because you have a student that's eager to get experience, and they need it for their portfolio. You need to invest in them, and then they will give back to the open source community, and all will benefit from that. Within the open source community, collaboration is very easy. It's very natural. I am very proud of our collaboration with Red Hat. We have a very fruitful collaboration in very different aspects. We have a very successful project from a master's student contributing to the CEPH open source project (bit.ly/cephobjectstorage), and the mentor was from Red Hat, for example.

We also have workshops on open source that started during the summer semester of 2020, when Red Hat's Beyond platform offered a class on open source development in conjunction with the IDC (see "Combining experience with passion," in this issue). The goal was to give undergraduate students an inside taste of development practices in the industry. The workshop was very successful, and we received thank you letters from students who appreciated the investment from the organizers and from the Red Hat engineers who participated as mentors.

Idan Levi: A master's student at IDC is doing a thesis with the cooperation of Red Hat. The subject is "Kubernetes optimized service discovery across clusters" (bit.ly/kubernetesclusters). What can you tell us about



The **Submariner project** provides an ability to connect multiple Kubernetes clusters into a secure shared network that allows various services to communicate with each other. Currently, the services can discover each other in a very rudimentary way using internal DNS queries. This research project aims to provide better and more balanced service discovery capabilities for such multi-cluster deployments. The project is exploring both proven and new techniques to allow a better service discovery experience, one that takes into account the cost of different paths, cost of services, and other parameters when recommending which services to use across the multiple clusters.

the background of how this work came to be and how it's going?

Anat Bremler-Barr: The student, Daniel Bachar, is doing the project under the supervision of Prof. David Hay from the Hebrew University and myself. He has and will continue to contribute code to the Submariner (submariner.io) project.

Kubernetes can be combined into different clouds and different clusters, and the same service can be found in different places in different clusters. Today, the way they choose, because applications today are done as microservices, you go from one service to another service. And then you choose whether to use the next service because you have it in a different cluster. If it is in your cluster, you probably will choose it. But if you need to go out, then you should think about the cost and the latency. We try to suggest an algorithm that will optimize both the cost and the latency. We wanted to combine it with DNS so it will be integrated very, very simply, without many changes in the framework.

Idan Levi: A big part of this project has been done in conjunction with the open source community and the SIGs. This is a new approach—at least, it's the first time that I have seen something like that. What was that like?

Anat Bremler-Barr: For me it's fascinating because one of the problems, I think, in academia is that you think about an algorithm, but then you must show it works. So you do a simulation, but you cannot implement it in real life. Now we have a chance to implement it in real life. It's a big problem, because it gives an advantage to people that have worked in big companies like Google or Microsoft. They have a network that they can play with and show results. Those of us in the university don't have a big network to play with.

So for me it was very effective. The fact that we can implement it in an open source project is very attractive. It's a win-win situation. And from another perspective, you see that the big companies attract many students and employees. We have more and more problems finding teachers and teaching assistants and staff for our applied computer science courses. The theoretical experts we can find. The problem is the applied computer science students can earn a lot of money in the industry. In some sense, the industry is chopping down the tree they are sitting on.

Idan Levi: It's like overfishing. When you catch all the little fish, they don't reproduce. You don't have teachers of tomorrow to teach the next generation.

Anat Bremler-Barr: Yes. For example, when I've been looking for a TA for a course, many people on staff tell me that their employers won't let them go to classes. In the past a business might say, "It's good for you—you can increase your understanding and develop as an employee." Now they do not.

Companies want to educate their employees on their own, but I think each of us should stick to what we know best. Leave the teaching to the university, but collaborate with us, help us, and be part of us. Fund us, give us projects—that's great. But you should not get your education from industry because they don't think about the welfare of the student. They are focused on the welfare of the company.

Really, to do great stuff, it takes a lot of effort and collaboration. Collaboration should be the default.

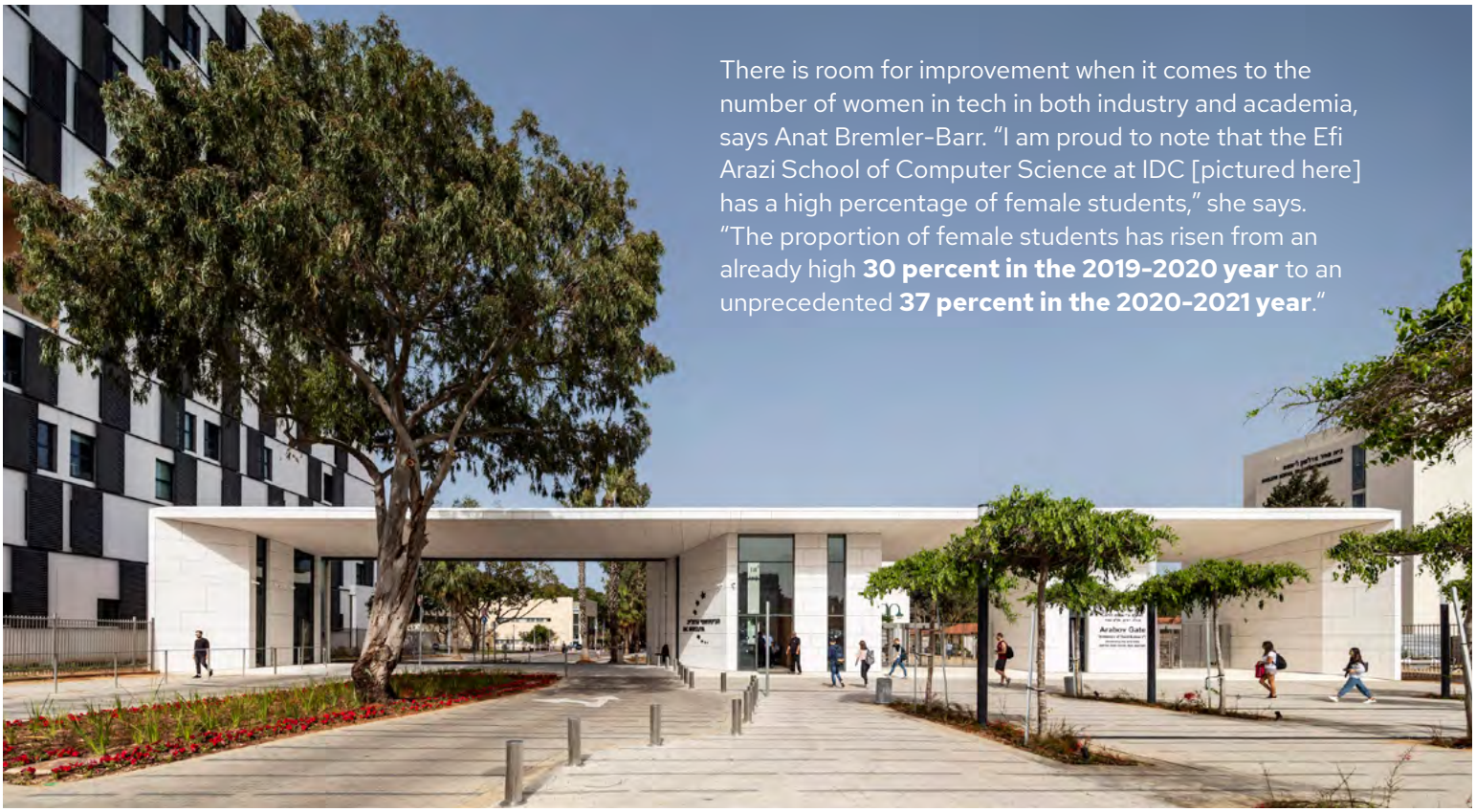
Idan Levi: So what would you recommend that industry do to make this collaboration better? You've been on both sides of this, as the founder of a start-up company that was acquired by Cisco as well as an educator.

Anat Bremler-Barr: Well, part of the problem is also the universities. We are very different at IDC, but at many Israeli universities the rules around intellectual property (IP) are very complex. The bureaucracy can be stifling. In industry, I think the culture needs to change, and more emphasis needs to be put on collaboration. I see a little bit of change already. Every year I do a workshop with the industry. Companies come and bring projects to the students. Some companies start to bring me lawyers. I tell them, "I cannot do this with lawyers; they complicate the situation." IDC does not take any IP. It belongs to the student, and the student can allow the company to use it. That's it. I don't want to take anything from the student. The student is doing the project, and I want the company to use it. I cannot read all the contracts and then pay our lawyers to speak to their lawyers and nothing gets done. No. I want it simple and collaborative. Really, to do great stuff, it takes a lot of effort and collaboration. Collaboration should be the default.

Idan Levi: Just a note here: Red Hat might be unique in the approach it takes to IP. We make no claims to IP when we work with researchers and students, as long as it's open source. But back to industry: In many cases, industry focuses on understanding what customers need. Is it security? Ease of use? Automation? How does being customer-centric translate into your world?

Anat Bremler-Barr: Well, we in academia have the privilege to think about problems in a fresh way. We can make radical suggestions and radical changes if overall, in the long run, they are better. In this sense, we are different from industry, which needs to think short term and focus on their paying customers. It does not mean that we do not think about the customers, but we can also concentrate on long-term satisfaction.

Idan Levi: Can we conclude with your observations on increasing the participation of women in computer science? As a woman in technology with a significant career, you are a role model and, I've heard, an inspiration to other women in the field. What do you think about the way that women are included in the academic world and industry?



There is room for improvement when it comes to the number of women in tech in both industry and academia, says Anat Bremler-Barr. "I am proud to note that the Efi Arazi School of Computer Science at IDC [pictured here] has a high percentage of female students," she says. "The proportion of female students has risen from an already high **30 percent in the 2019-2020 year** to an unprecedented **37 percent in the 2020-2021 year**."

Anat Bremler-Barr: It is clear that there is room for improvement in the number of women that are included in academic work and the industry. Though when I entered the Red Hat office in Ra'anana, I was amazed to see the high percentage of women, and it struck me to see that there were women in all the ranks in all ages. Unfortunately, this is not the typical case.

I am proud to note that the Efi Arazi School of Computer Science at IDC has a high percentage of female students. The proportion of female students has risen from an already high 30 percent in the 2019-2020 year to an unprecedented 37 percent in the 2020-2021 year.

The increase can be attributed to two main factors. First, it reflects a general trend around the world, with increasing numbers of women studying the computer sciences, attracted at least in part by the availability of well-paid jobs in the field. And second, it reflects a determined push by the Efi Arazi School and IDC Herzliya in general to encourage more female students to enroll in computer science. We hold events annually targeted towards encouraging women to join the school, and an active recruitment program continues throughout the year. In addition, 30 percent of faculty members are women, a significantly higher percentage than in other universities (around 15 percent in Israel), which helps attract more female students.

That said, I do have a hard time finding a female TA in operating systems. For 20 years I didn't have a female TA. Sometimes I begged! "Please come. You will be so perfect as a TA." The last one I had was 20 years ago, and that is a shame, really a shame.

Red Hat

RESEARCH QUARTERLY

VOLUME 3:1

Feature

Verifying programs that communicate with the environment

Writing tests with high coverage is almost always tedious work that is still error prone. This can lead to missing crucial details that cause undesirable behavior, and, in the worst case, a complete system failure. What if there were an efficient way to automate this work?

by Henrich Lauko

About the Author

Henrich Lauko is a PhD candidate at the Faculty of Informatics, Masaryk University in Brno. He is intrigued by both theoretical and engineering aspects of computer science. He finds the sweet spot for both in program verification, where he focuses on the development of reusable analyses using compiler toolchains.

In my research, I investigate how to automate testing through a systematic exploration of all admissible program inputs. The technique I have developed is called *compilation-based abstraction*. In this technique, the program under test is transformed to compute with a set of input values instead of a single value. This allows exploration of multiple program inputs (possibly all) at once, hence providing higher guarantees than just pure testing.

HOW TO TRUST YOUR CODE?

In general, to gain trust in your code, you write tests that simulate interactions with the environment; that is, you capture concrete scenarios with encoded input values. Alternatively, you can deploy more sophisticated analyses, like fuzzing or randomized testing. However, the reliability of these techniques is influenced by a human factor. The developer might always miss some test cases or omit some properties of the code entirely.

The natural way to mitigate the human factor is automation. Generating test cases automatically is an appealing solution; however, examining all possible program interactions presents a problem. If we would like to explore all possibilities naively, even for a simple program that only takes a single integer as an input value, that would already make 2,147,483,647 possible executions to explore. However, many of those executions take the same path. In automation, we would like to generalize them. In fact, developers also apply a similar process when they design test case scenarios.

Examine the artificial program in **Figure 1**. Can you guess for which values of *a* and *b* the assert (post-condition of the procedure) will be triggered?

Examining all possible inputs is indeed unnecessary. We may recognize that there are only a few classes of inputs in which the program behaves differently. For example, for all negative inputs of variable *a*, the procedure's behavior is always the same because, in all executions, we perform the same instructions (we do not execute the code inside the block of the first condition).

```
1 void procedure() {
 2   int a = input();
 3   int b = input();
 4   int x = 1, y = 0;
 5   if (a > 0) {
 6     y = x + 3;
 7     if (b == 0)
 8       x = 2 * (a + b);
 9   }
10   assert(x - y != 0);
11 }
```

Figure 1. *Artificial program*

For efficient automation of exhaustive testing, we want to find a way to describe these classes efficiently and perform computation with them. One such technique that allows us to compute with sets of values is symbolic execution.

SYMBOLIC EXECUTION

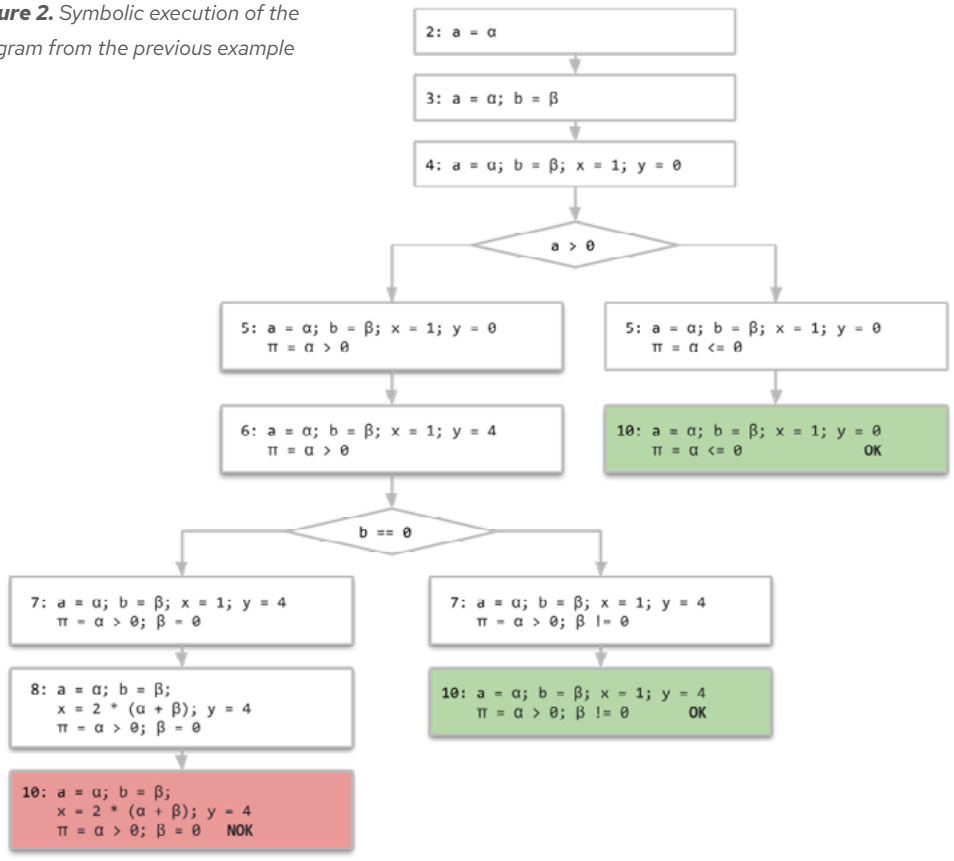
Symbolic execution is a program analysis based on the same idea as testing: it executes paths in a given program to find bugs. However, symbolic execution and testing differ in how they actually execute program paths. In testing, we execute a given program for each input in a given set. This means that the program is passed many inputs during testing, and for each input, the corresponding execution follows exactly one program path. On the other hand, in symbolic execution, the program is passed only one input. This single input does not consist of concrete data (like numbers or strings); rather, it is represented by symbols (names of inputs). These symbols represent any concrete input data we can possibly pass to the program. The program is then executed with these symbols. Individual program statements generally manipulate expressions containing symbols instead of performing common calculations with numbers.

Figure 2 shows all possible paths of symbolic execution of the program from the previous example. Boxes describe values of variables at each program location. Each location also contains a so-called path-condition π that keeps all the constraints on the symbolic values. The symbolic values used in the example are α and β.

Notably, symbolic execution gives us the following guarantees:

- It executes only truly executable program paths, i.e., those which can be followed for some concrete input by standard execution.
- Each executable path is symbolically executed at most once.
- For a symbolically executed program path, we can directly compute a representative concrete input for which standard execution will follow exactly that path.

Figure 2. *Symbolic execution of the program from the previous example*



Even though the symbolic execution technique seems promising, there are scenarios in which it is too slow to be of any use. This is caused by expensive computation with symbolic expressions and a so-called *path explosion problem*, i.e., when an enormous number of paths is generated during the execution. To mitigate these problems, we can leverage techniques like *abstract execution* (interpretation), but we pay the cost of analysis precision (coverage). In comparison to symbolic execution, the abstract execution does not compute with symbols but rather with an abstract representation of values.

Abstract representation describes only some properties about values, for example, whether the value can be a null pointer, signed integer, or a particular form of string. We gain execution performance by abstracting only specific properties, since a single abstract path might describe multiple symbolic paths. However, this is in trade for the precision of analysis. By keeping track of only specific properties, we might omit some paths that lead to an error location, which we call underapproximating abstraction. Or, in the case of overapproximating abstraction, we might find an error that is unreachable in real execution, i.e., a false positive.

In abstract execution, we pick these properties beforehand in the form of the domain in which we want to compute. This domain then describes admissible abstract values. For example, to track the nullity of pointers, we may use a simple domain that consists of values: is zero, is nonzero, or is unknown (can be both zero or nonzero).

In general, we can look at both symbolic and abstract execution like they are performing an execution with sets of values instead of a single concrete value. For example, a nonzero value from a nullity abstract domain represents a set of integers {1, ..., MAX_INT}, whereas the same set of values in the symbolic domain might be described as the expression x > 0.

What differentiates the abstract and symbolic execution from normal (concrete) execution is the ambiguity of control flow. Imagine you have a symbolic value *v* described by the expression x > 0. If this value is used in branch condition v > 10, both outcomes are admissible, either v > 10 or v <= 10. To deal with this situation, a symbolic executor needs to explore both paths separately.

COMPILATION-BASED APPROACH

In computer-aided verification, most of the tools leverage abstraction techniques to reduce the complexity of analyzed systems. Even though these techniques are widely adopted, they are usually tightly integrated into tools: abstract semantics are an internal part of interpreters. This causes undesired complexity and neglects any reusable design. In my research, I devise a self-contained alternative to perform previously described abstractions independently of the tool. This self-contained approach is called compilation-based abstraction.

In the traditional approach, the abstraction execution is implemented in the interpreter. As you might guess, the compilation-based approach performs the abstraction before the actual analysis, during compilation. In short, the abstract compilation transforms program instructions that operate with user inputs to work with abstract representation instead, such as symbolic expressions. Once the program has been transformed in this way, it can then be analyzed by an arbitrary verification tool. The only requirement is that the tool (interpreter) can explore ambiguous branching because we do not know yet how to encode branching execution directly into the program simply. But in this way, the verification tool does not need to know about abstraction. In the big picture, the verification workflow with compilation-based abstraction looks like the diagram in **Figure 3.**

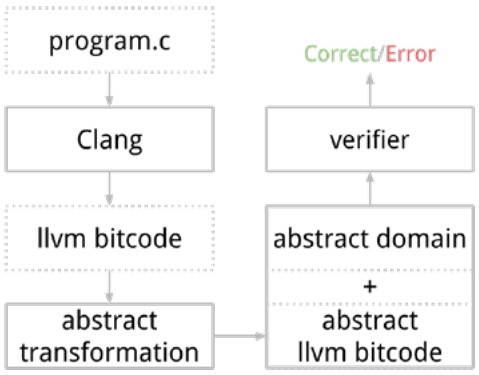


Figure 3. *Verification workflow with compilation-based abstraction*

By comparison, the traditional workflow would implement and perform interpretation directly in the verifier. The compilation-based approach clearly offers some advantages. First of all, the simplified verifier is more efficient and more resilient to errors, giving us higher guarantees of the verification results. In fact, the verification also verifies the actual abstraction. On the other hand, the transformation increases the size of the analyzed code that needs to be interpreted, impacting the interpretation of the bitcode, which is, in consequence, slower than the direct implementation of abstraction in the verifier.

The solution I propose is implemented on the level of LLVM bitcode, which is simpler for analysis and transformation than the original C/C++ code. I have prototyped a solution in the DIVINE verifier. It is able to interpret LLVM bitcode and explorer branching executions. However, before integrating the compilation-based method, DIVINE could not process programs that take user inputs. The part of DIVINE that performs transformation is called LART: LLVM Abstraction and Refinement Tool.

BUILDING A VALUE ABSTRACTION

One of the design goals for compilation-based abstraction is to be accessible for developers, so that it is easy to create its own value abstractions. This is mainly achieved by implementing the actual abstract domain as a C++ library. To show you the ease of domain creation, let us now implement the previously mentioned nullity tracking domain, also called *zero domain*.

The implementation of an abstract domain is realized as a C++ class that defines how abstract values are represented and provides available operations on this representation. In the case of the zero domain, the representation is particularly simple: each value just maintains the information about its nullity. We represent the nullity information with a single enumeration of possible value states:

```
struct zero_domain {
  enum value {
    zero,
    nonzero,
    unknown
  };

  ...
```

The value can be either zero, nonzero, or unknown if the nullity is undetermined. An abstract value in the program can then be created either by an abstract input function that simulates an arbitrary input from the environment, or by lifting a concrete value:

```
value input() {
  return value::unknown;
}

value lift(int i) {
  if (i == 0)
    return value::zero;
  return value::nonzero;
}
```

In short, the abstract compilation transforms program instructions that operate with user inputs to work with abstract representation instead, such as symbolic expressions. Once the program has been transformed in this way, it can then be analyzed by an arbitrary verification tool.

Given this value representation, the domain then defines arithmetic operations:

```
value add(value a, value b){
  if (a == value::zero)
    return b;
  if (b == value::zero)
    return a;
  return value::unknown;
}
```

For example, in the addition, we check whether one of the arguments is zero. In such a case, the result is the other value because the addition of zero does not change the result. In other cases, the result can be anything (an unknown value) since the addition of two nonzero values can result in both zero or nonzero due to integer overflow. The addition to unknown value is also undetermined.

Besides arithmetic operations, the abstract domain also needs to

implement relational operations. For example, an equality operation (eq) in the zero domain can be implemented as in the following snippet. Note, we keep the semantics of the C language, where integer values also represent boolean values, so the result of comparison in the zero domain would also be a value in the zero domain.

```
value eq(value a, value b) {
  if (a == value::zero) {
    if (b == value::zero)
      return value::nonzero; // true
    if (b == value::nonzero)
      return value::zero; // false
  } else if (b == value::zero) {
    if (a == value::zero)
      return value::nonzero; // true
    if (a == value::nonzero)
      return value::zero; // false
  }
  return value::unknown;
}
```

The comparison is ambiguous (unknown) in the case when we compare two nonzero values; the abstraction is too coarse to determine equality of nonzero values. In the case when a program is to perform a branch based on an unknown value, we need to split execution and examine both paths, as shown in the introductory example.

ABSTRACTIONS OF DATA STRUCTURES

Besides numeric values, we often manipulate programs with more complex data: arrays, strings, and other data structures. These are also interesting candidates for abstraction since many of those can come from the environment, and we want to explore multiple executions at once. The abstraction of data structures does not principally differ from numeric abstraction. In the domain, you describe an abstract representation of a data structure and implement its operations.

An example of such an abstraction would be a string abstraction suited for the verification of C programs, which I have developed during my research. This particular abstraction allows us to represent infinitely long strings with abstract characters as well. It is designed to repurpose numerical domains to represent characters and the length of strings. Moreover, the abstraction leverages the fact that the compilation-based approach can also abstract whole functions. Hence the string domain may provide its efficient abstract implementation of standard C library functions like **strcmp**, **strcpy**, etc.

Bitcode transformation also has uses beyond value abstraction. It provides the capability to perform arbitrary computation on the execution of instructions and functions. For example, one can implement statistical analyses that count the number of executions of particular instructions. Furthermore, the domain operations can check for specific properties during runtime. For example, we can utilize the transformation for security analyses and check whether some user input is used in a forbidden computation (e.g., memory manipulation).

CURRENT RESEARCH

As many abstract domains are imprecise, my current research focuses on domain refinement. The general idea behind domain refinement is to detect whether the found error is a false positive. In that case, the abstraction has to be augmented to forbid a particular error location's reachability. In a compilation-based approach, this can be done with a simple swap of the abstract domain; the transformation does not need to be repeated. The only thing we need to do is to link new semantics (domain) and rerun the verification.

The other portion of my research focuses on the integration of multiple domains in a single program. This is a particularly challenging problem, where one needs to solve what to do when values from different domains come as arguments to a single operation.

The implementation of the transformation tool LART and domains presented in this article are currently a part of the verification framework DIVINE: divine.fi.muni.cz. However, there is also a work in progress on a standalone implementation at my GitHub repository: github.com/xlauko/lart. RH RQ

RESEARCH QUARTERLY

VOLUME 3:1

Red Hat

Feature

Translation layers for the cloud: speeding storage performance

A guide to understanding the hidden algorithms that manage the data in our everyday world, from smartphones to cloud apps. We look at which ones perform faster—and why.

By Peter Desnoyers

Block translation layers handle much of our data. These algorithms are hidden away inside our SSDs, the storage in our phones, or the systems that store Dropbox files from a few years ago that you've probably forgotten about. They transform difficult-to-use devices like NAND flash or shingled disks into well-behaved ones, supporting the rewritable block interface that our file systems know and love. But these translation layers are good for something besides making weird chips and disks safe for file systems. We're using them in the cloud, creating virtual disks on top of S3 object storage. To explain why, we'll take a detour through translation layers, S3 object storage, and disk consistency models first.

We're using them in the cloud, creating virtual disks on top of S3 object storage.

A block translation layer provides a simple rewritable block interface over something else that doesn't. The most widely known examples are Flash Translation Layers (FTLs), used almost everywhere flash memory is used. NAND flash itself is difficult to use: although it's divided into pages about the size of a disk block, and the pages can be read independently, the similarities end when we get to writes. Hundreds of these pages are grouped into erase units, and the pages in a unit must be written one at a time, in order, until it's full and can't be rewritten until all of them are erased in a single operation. A flash translation layer accepts disk-like block read and write requests, and uses out-of-place writes, a dynamic translation map, and garbage collection to implement them on top of flash. Since it can't overwrite existing data, each write goes to a new location and updates a logical-to-physical map used for reads. The old location is now invalid—i.e., garbage—and when enough garbage accumulates, the remaining data is copied out of an erase unit so that it can be erased and made available for new writes.

But why is S3 storage like NAND flash, and why does it need a translation layer? At first glance S3 looks like a file system: variable-length objects have names and hold data, and reads are allowed

About the Author

Peter Desnoyers is an Associate Professor in the Khoury College of Computer Sciences, which he joined in 2008. He is one of the founders of the Mass Open Cloud, a multi-institutional collaboration to develop new models for cloud computing, and serves on the steering committee. His research is focused on storage issues in operating systems, in particular the integration of emerging storage technologies such as flash and SMR disk into existing software infrastructures.

So why are we going to so much trouble to create a virtual disk over S3? The answer comes back to local caching, and in the end to yet more issues of consistency.

with arbitrary byte offsets and lengths. Like flash, however, writes are different. S3 objects must be written in a single operation,[1] which either creates a new object or replaces an existing one. Once you've written an object you can read it or delete it, but the only way to modify it is to rewrite the entire thing. Why is S3 so limited? In a nutshell, because it's easier that way,[2] because of issues of replication and consistency.

CONSISTENCY VS. PERFORMANCE

Cloud storage systems like AWS S3 or Ceph replicate data on multiple machines for reliability, and they are designed to transparently handle failure of any of these machines without affecting the user. One of the hardest parts of this failure handling is keeping these replicas consistent with each other. For instance, if replica A is down when I make a change to replicas B and C, but then comes back up, I risk getting different data on each read depending on which replica it is routed to. Avoiding this requires mechanisms like locks and write-ahead logs, which add complexity and subtract performance, and it gets even worse when you go from simple replication to erasure coding. In contrast, when you create a new write-once object, consistency is simple: if you have a copy of it, then you know it's the right one. Overwrites are a bit trickier, but not by much, mostly because S3 makes very few promises about when you'll actually see the new copy.

In the open source world, Ceph is the most widely used cloud-scale storage system, and it supports both write-once and rewritable abstractions. At the lowest layer it has a pool of Object Storage Devices (OSD) storing rewritable objects; unlike S3 objects, these really do work like files. The Ceph RADOS Gateway (RGW) provides an S3 object service over these OSDs, splitting large S3 objects into smaller fixed-sized Ceph objects, writing multiple smaller objects in parallel for higher throughput. Although OSDs provide mechanisms to modify these objects safely, RGW never uses them, and so never pays the performance price of write-ahead logging and other mechanisms for preserving consistency.

The Ceph virtual disk, RADOS Block Device (RBD), takes advantage of Ceph's rewritable objects by splitting a virtual disk image into smaller fixed-size Ceph objects, and translating disk block reads and writes into reads and writes of the corresponding object byte ranges. In contrast, creating a virtual disk over S3 requires something that looks a lot like a flash translation layer: new writes go to new S3 objects and update a translation map, and any remaining live data in old S3 objects is garbage collected before the object is deleted.

So why are we going to so much trouble to create a virtual disk over S3? The answer comes back to local caching, and in the end to yet more issues of consistency. As high-speed NVMe drives become more and more affordable, it becomes very tempting to use a local cache for virtual disks, as these local IOPS are far cheaper than

equivalent performance in a shared storage cluster. Unfortunately if you do this wrong, the price you pay might be your file system.

Modern file systems are crash consistent: they order their writes in such a way that the file system is unlikely to be corrupted by a crash,[3] typically by using a write-ahead log or journal for metadata updates such as directory entries and allocation bitmaps. To achieve this ordering, while still using asynchronous writes for high performance, file systems (and the fsync system call) use commit barriers—operations like the SCSI synchronize cache command, which guarantee that all preceding writes will be performed before any following ones.

A simple local cache with asynchronous write-back (there are three available in the kernel, and several other ones) can easily be combined with a virtual disk such as RBD, resulting in tremendous boosts in performance. Everything will be fine as long as neither the local SSD nor the virtualization host itself fail permanently, but if they do, things get messy. When this happens all that's left is the remote virtual disk image.[4] The bcache documentation describes the likely result for a cache that ignores commit barriers: "you will have massive filesystem corruption, though ext4's fsck does work miracles."

One alternative is to use a write-through cache, but that sacrifices much of the speed advantage of a local cache. The other alternative is to use a cache that preserves commit barriers, several of which are described in the literature. This second approach works great for workloads with few commit barriers; some that we've measured in the lab have one or two barriers per gigabyte written, and would be unaffected. Other workloads (SQLite is a prime example) send commit barriers writes after every few writes to the disk, and show little improvement over simple write-through.

By using a translation layer we sidestep this problem entirely. After logging writes to local SSD for durability in the case of recoverable crashes, our system batches a sequence of writes into a single object, gives it a sequence number (embedded in the object name), and writes it to the back end. If multiple object writes are outstanding when the system crashes, it's possible that a random subset of them will fail to complete. However unlike RBD (or iSCSI, QCOW2 over NFS, etc.), we still have the old data, and we can decide which updates to apply and which to discard. In particular, we examine the sequence numbers and find the first gap: all updates before this gap are applied to the volume, and any objects following that sequence number gap are discarded. This preserves commit barrier semantics: if we keep a write following a commit barrier, then any write before that barrier is either in the same object, or in a preceding one that is guaranteed to exist by our recovery rule.

We've implemented a prototype of a translation layer over S3, split into a kernel device mapper and a Golang-based user-level daemon, and are testing it extensively. We hope to deploy a version of this as a pilot storage pool in the Mass Open Cloud (massopen.cloud) later this year.

---

[1]Or a multi-part upload, but the result is the same.
[2]In engineering, "easier" often means "cheaper", "more reliable", or even just "possible."
[3]Having used Linux since kernel 1.0 and the ext2 file system, I can attest that this was not always the case, much to my occasional distress.
[4]Note that most of these caches were designed to cache local hard drives, where this scenario was unlikely.

Feature

Combining experience with passion inspires a new mentorship program

The Office of the CTO is promoting open source development concepts among teenagers, the military, and the academic world in Israel. Here's how two engineers helped realize that goal.

by Irit Goihman and Liora Milbaum

About the Authors

Irit Goihman is a software engineering manager at Red Hat and an open source enthusiast.

Liora Milbaum is a senior principal software engineer at Red Hat with a passion for DevOps culture.

Red Hat Beyond is an initiative that gives students a taste of real-world software development and methodologies, by exposing them to the entire process of designing and developing a software project in a collaborative environment. Students have an opportunity to combine the theoretical knowledge being taught in the academy with hands-on experience with a real-world tech stack.

In the program, professional engineers prepare and present weekly sessions for the students and take responsibility for mentoring them. Mentors work with students to understand what is involved in real-world software development using an open source model, and they answer questions about technical difficulties. Students are also exposed to using GitHub, the code review process, and coding style requirements.

Students are divided into teams of five, each one taking on a web application development project to complete. As part of the project, each team had to design the system architecture, database, and front end screens in addition to the back end in Python. While the Red Hatters are there for support and direction, the students themselves own the entire project. To this end, they learn how to collaborate with each other and divide their assignments so each team member has an area of responsibility.

A HISTORY OF LEADERSHIP

The Beyond initiative was started by Red Hat associates Liora Milbaum and Irit Goihman. Liora, a senior principal software engineer, joined Red Hat after twenty years of running a company providing DevOps services. In her company, Liora trained junior engineers and gave them the tools to start their own journey in the DevOps world.

Training younger people was one of the things Liora felt fulfilled her the most. To continue this work, she started the DevOps Loft initiative, a nonprofit community for aspiring DevOps engineers willing to gain more knowledge and tools to bootstrap their careers. She ran weekly meetups where senior DevOps engineers provided workshops and shared their knowledge with others.

When Liora joined Red Hat, she discovered the company's efforts to connect the industry and the academic world and suggested applying the

methods she used in DevOps Loft in the university. Her proposal included creating a different kind of academic course, with real-world, hands-on experience for students. This proposal was warmly welcomed by Sofi Sherman, PhD, and Prof. Ruti Gafni from the Information Systems school of the Academic College Tel Aviv Yafo.

Irit, a software engineering manager, has ten years of experience in the industry, working in R&D as well as IT. She is passionate about sharing her knowledge to help other women succeed in the engineering field and promoting diversity inside and outside of Red Hat. Among her projects is a school visit program at the Tel Aviv site, in which high school students spent full days at Red Hat and learned about open source culture.

Liora and Irit partnered to lead the Beyond initiative together. The program was a perfect opportunity to mesh Liora's and Irit's experience and passion while passing on knowledge and skills to future engineers early in their careers. The tech industry recruitment process puts more emphasis on relevant experience and skills than on degrees, which can create challenges for less experienced candidates. A program like Beyond helps potential engineers gain more background knowledge and leverage open source projects as additional experience.

CONTINUED GROWTH

When Irit and Liora presented their Introduction to DevOps course in the Information Systems school, they met Prof. Gideon Dror, the Computer Science Dean, in the elevator. Prof. Dror was eager to hear more about the course and immediately asked about incorporating a similar course in the computer science department as well. This is how the second course, Open Source Development Principles, was initiated.

During the second course, the COVID pandemic hit the globe. Beyond had to become fully virtual. This was a surprisingly easy transition. as the program members used the same tools that Red Hat engineers use for their daily work in a global environment.

Another virtual class about open source development was offered in conjunction with Prof. Anat Bremler-Barr, Vice Dean of the Efi Arazi School of Computer Science at the Interdisciplinary Center (IDC) of Herzliya, Israel's only private university.

So far, four courses have been delivered in two different academic institutions. Additional workshops have been offered at Haifa University, Bar Ilan University, and for the Israeli Navy.

The success of the program can also be seen in the organic spread of open source ideas. For example, after completing an open source course with Beyond, an IDC computer science student started working on a project with a few friends from the army and was able to teach them open source software development methods. In this way, Beyond has created new opportunities for collaboration and the growth of open source culture well after the course is over. RH RQ

WHAT THEY'RE SAYING

"The course challenged the students and demanded a lot of self-learning. But eventually it was a big boost both to their technical knowledge and their soft skills.

—Prof. Gideon Dror, The Academic College Tel Aviv Yafo

"Nowadays, when industry is advancing extremely fast towards new and exciting technologies, maintaining a strong link between industry and academy is absolutely vital.

—Sofi Sharman, PhD, Prof. Ruti Gafni

Red Hat

RESEARCH QUARTERLY

VOLUME 3:1

Column

Planting the seeds for a blossoming research program

How Czech Technical University and Red Hat Czech broke ground for grant programs.

by Matej Hrušovský

About the Author

Matej Hrušovský has been with Red Hat for more than seven years, five of which have been spent managing the university program in EMEA. Aside from attracting new talent mainly from universities and schools, the core of Matej's job is to find and put the right people from Red Hat and academia in the same room together.

It all began in 2017. The university program in Brno was thriving, and collaboration with Czech Technical University (CTU) in Prague was just beginning to gain traction. Red Hat Czech had been on the receiving end of a couple of Czech government subsidies to support research work but never connected to an actual research grant. CTU took the initiative that got our joint research efforts started, setting a pattern for university-industry-government collaboration that continues today.

When the cooperation began, we wanted our software engineers to know more about what the researchers at the Faculty of Electrical Engineering of CTU were doing to expand our knowledge of the latest innovations. Since Prague and Brno are more than two hours apart, which makes continual collaboration much harder, we invited them to Brno to speak about their research. This sparked a lot of interest, and the presentation was not only well attended in person—which sounds a little otherworldly these days—but also streamed to those who couldn't be there. It was at this presentation that the conversation between the academic researchers and the Middleware Quality Engineering (QE) team began.

> It was the first of its kind for us, but it has opened the door for two more projects that are still ongoing.

A few months later, representatives of the Faculty of Electrical Engineering at CTU contacted us with a proposition. They discovered a Call for Proposals (CFP) from the Technology Agency of the Czech Republic (TACR), and the university needed an industrial partner to apply for a three-year cash grant program. The project that resulted from the conversation with the Middleware QE team was a good fit. We agreed with the university to propose the Quality Assurance for Internet of Things Technology project for the grant.

The application was the first major hurdle. Being a partner for such a project was unprecedented at Red Hat Czech, and there was no approval process in place. Finding the right people to talk to, then convincing them to invest time into reviewing and giving the green light to a project that was somewhat insignificant from a broader business point of view was an exercise in perseverance. Thus it was all the more rewarding when the project won the selection process.

Red Hat

Fast forward to 2021: the project concluded in December 2020 and it has found a lot of success in both the academic and the industrial fields. When it comes to PatrIOT, Red Hat's part of the project, two out of three patent applications have already been accepted. Moreover, while the grant program has concluded, the project is not finished. The partners' cooperation will continue in 2021, self-funded for now, while seeking a new CFP to enable further collaboration.

The framework has been applied in several projects, including Red Hat AMQ and a prototype IoT-based rescue mission and planning system for the Czech police and mountain rescue service. Multiple institutions—Johns Hopkins University, NATO ACT Innovation Hub, CTU, Armed Forces of the Czech Republic, and DefSec Innovation Hub—have used the framework to develop an experimental sensor network for monitoring soldiers' vital functions that facilitates more accurate triage and minimizes casualties.

Quality Assurance for Internet of Things Technology has been our icebreaker project that brought academic researchers, industry software engineers, and a government agency together. It was the first of its kind for us, but it has opened the door for two more projects that are still ongoing. Red Hat has become more active in applying for grant programs, and getting the applications through has now become much easier thanks to an existing precedent and tangible results that the projects bring to research and business. RH RQ

Research project updates

Here are a few highlights of recent research results from the US. There are many more active projects than we can cover here, so be sure to check research.redhat.com listings for additional projects. We will highlight research collaborations from other parts of the world in future editions of *RHRQ*. Contact academic@redhat.com for more information on any project.

You can join live Research Interest Group (RIG) meetings each month to discuss new project proposals and review the latest results from other research collaborations. Subscribe to the US-Research@redhat.com mailing list to stay current on the interest group meetings.

PROJECT: FPGAs in Large-Scale Computer Systems

ACADEMIC INVESTIGATORS: Martin Herbordt, Robert Munafo, Orran Krieger, Rushi Patel, and Mayank Varia (Boston University)

RED HAT INVESTIGATORS: Ulrich Drepper and Ahmed Sanaullah

Investigators on this project recently moved closer to their goal of enabling FPGA (Field Programmable Gate Arrays) application development by high-level language programmers, especially those working in datacenter and edge environments, using only open source tools. They demonstrated a hardware implementation of secret sharing using FPGAs and assessed the scalability of the design against comparable software-only implementations. Their results, shared in a paper presented at the 30th International Conference on Field-Programmable Logic and Applications (FPL 2020) by Pierre-François Wolfe, are the first-ever results reported for secret sharing multiparty computation (MPC) on FPGA hardware (see bit.ly/MPConFPGAs).

MPC facilitates shared utilization of datasets gathered by different entities by enabling data from several sources to be used in a secure computation. Only the result is revealed, while the original data is protected. The presence of FPGA hardware in datacenters can provide accelerated computing as well as low-latency, high-bandwidth communication that bolsters the performance of MPC and lowers the barrier to using MPC for many applications. The group's most recent work demonstrated that secret sharing outperformed state-of-the-art methods for implementing MPC in the datacenter. Using 5.5% of FPGA fabric in a consumer cloud environment, this result can match

the throughput of an optimized 20-core CPU implementation, saturating a typical 10Gbps network connection. This result scales with available bandwidth: a single FPGA is able to saturate a 200Gbs link with a throughput of ~26 million AES operations per second.

PROJECT: Kernel Techniques to Optimize Memory Bandwidth with Predictable Latency

ACADEMIC INVESTIGATORS:
Parul Sohal, Renato Mancuso, and Orran Kreiger (Boston University)
Rohan Tabish (University of Illinois at Urbana-Champaign)

RED HAT INVESTIGATORS: Ulrich Drepper and Larry Woodman

Parul Sohal has presented a paper with her co-authors Rohan Tabish, Ulrich Drepper, and Renato Mancuso titled "E-WarP: a system-wide framework for memory bandwidth profiling and management" at the 41st IEEE Real-Time Systems Symposium.

The paper, which won the RTSS Best Student Paper award, used a profiling approach to model memory behavior and understand memory utilization with enough detail to predict application behavior under controlled conditions. As summarized in the paper, "Profiling represents a substantial refinement of measurement-driven approaches, where fine-grained knowledge of the interaction between applications and the platform is collected and leveraged. Conversely, we treat the DRAM subsystem, as much as possible, as a black box. By shifting our emphasis on a more precise representation of memory bandwidth requirements of applications and by ensuring that the DRAM subsystem operates below its saturation threshold, we demonstrate that highly accurate predictions on the behavior of tasks operating on CPUs and accelerators can be made." The E-WarP framework provides techniques to profile and bound the temporal behavior of application workloads on CPUs and accelerators, providing tools and details in two Github repositories. See bit.ly/kernelmemorylatency for more information on this project.

PROJECT: Open Telemetry Working Group

ACADEMIC INVESTIGATORS: Raja Sambasivan (Tufts University)

RED HAT INVESTIGATORS: Marcel Hild

OPEN SOURCE PARTNERS:
OpenInfra Labs, Open Cloud Testbed, and Mass Open Cloud

Red Hat and academic participants have been collaborating for some time to build, operate, and share infrastructure that demonstrates open source cloud operations at scale, most recently including the Operate First initiative at OpenInfra Labs (OIL). A significant step forward in this effort was the recent formation of the Open Telemetry Working group, with participants from several different universities, OIL, and Red Hat. The group seeks to build upon a realistic production-grade environment, operated by IT operations and used by end users and researchers alike. By exploring ways to provide access to telemetry data for research and open operations engineering in this environment, the group hopes to enable new research and development projects, in much the same way that the open source movement enabled new options for software development. Examples of research projects that would benefit from this type of environment include creating new debugging tools and visualizations, using telemetry data to optimize workload performance, and improving telemetry data itself. Monthly meetings are open to all interested participants. The group charter and working notes are shared publicly (see bit.ly/telenote) along with a Github repository (see github.com/open-infrastructure-labs/telemetrywg).

PROJECT: Deploying End-to-End, Fully Virtualized, and Open Source 5G Platforms on OpenShift

ACADEMIC INVESTIGATORS:
Tommaso Melodia, Abhimanyu Gosain, and Michele Polese (Northeastern University)

RED HAT INVESTIGATORS: Feng Pan

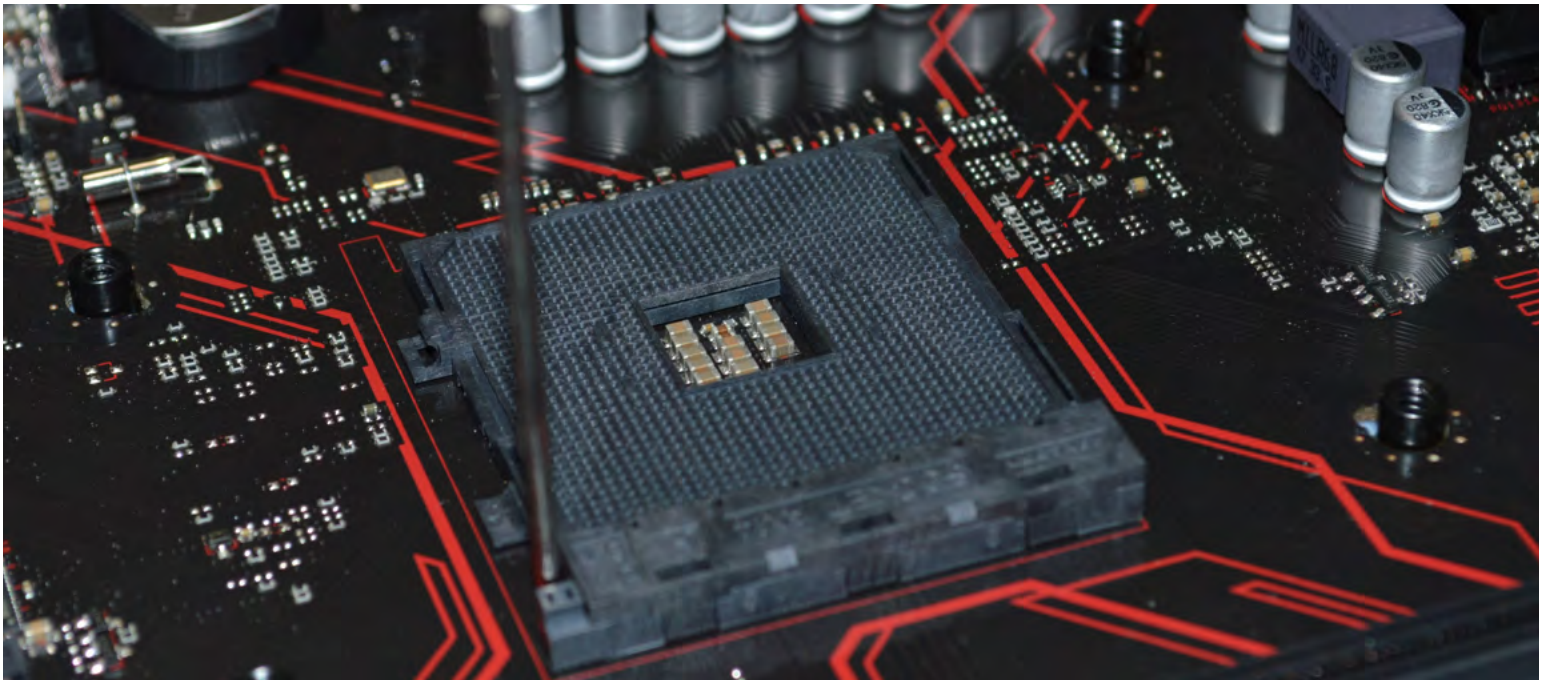
Traditional cellular networks are mostly based on closed source, inflexible architectures, in which functionalities are baked directly on hardware components (e.g., the base stations). This black-box approach leads to vendor lock-in

and is unable to adapt to the rapidly varying network, traffic, and topology dynamics that characterize 5G networks. This results in suboptimal network performance. In the last few years, a number of consortia, primarily led by telcos, have been promoting solutions to overcome this imposed lock-in by pushing equipment manufacturers to produce open hardware that can be (i) dynamically programmed via software and (ii) seamlessly integrated—through open interfaces—with a network architecture consisting of components provided by multiple vendors. The resulting network softwarization allows telcos to directly program algorithms and policies to optimize the network behavior in real time, based on the current conditions and requirements (e.g., traffic demand, Quality of Service [QoS], and latency), while opening the network to third-party vendors.

The goals of this new project are to develop an experimental open source platform that merges open and reprogrammable software and hardware components that can be used to test and deploy fully virtualized 5G networks. The platform builds on Red Hat OpenShift and large-scale national wireless experimental facilities:

- Arena—a 64-antenna SDR-based ceiling grid testbed for sub-6 GHz radio spectrum research
- PAWR—Platforms for Advanced Wireless Research, a National Science Foundation-funded program
- Colosseum—A massive radio-frequency (RF) and computational facility developed by the Johns Hopkins Applied Physics Lab to support the Defense Advanced Research Projects Agency's (DARPA) Spectrum Challenge

Ideally, the project will also connect to resources in the Open Cloud Testbed that can provide resources for building core network and datacenter testbeds. The project will develop automated pipelines using OpenShift to build, deploy, and manage these complex systems that combine radio, compute, storage, and networking resources into dynamic experimental testbeds that can cope with the tight real-time requirements of experiments with cellular networks. RH RQ



```
1995  const fetchingBlueprints = (state = false, action) => {
1996   switch (action.type) {
1997     case FETCHING_BLUEPRINTS:
1998       return true;
1999  // We went from selling boxes of Linux®
2000     case FETCHING_BLUEPRINT_NAMES_SUCCEEDED:
2001       // if 1 or more blueprints, fetching is true because we're still waiting
2002  on the contents)
2003       return action.payload.blueprints.length > 0;
2004     case FETCHING_BLUEPRINTS_SUCCEEDED:
2005   // to operating servers around the globe
2006   case BLUEPRINTS_FAILURE:
2007       return false;
2008     default:
2009       return state;
2010   }
2011  };
2012  const errorState = (state = null, action) => {
2013   switch (action.type) {
2014  // to giving new life to virtual machines
2015     case FETCHING_BLUEPRINTS:
2016     case FETCHING_BLUEPRINTS_SUCCEEDED:
2017       return null;
2018     case BLUEPRINTS_FAILURE:
2019       return action.payload.error;
2020  // to pioneering a new era in containers
2021     default:
2022       return state;
2023   }
2024  };
2025  // to powering over 90% of the Fortune 500*
2026  const blueprintList = (state = [], action) => {
2027   switch (action.type) {
2028     case CREATING_BLUEPRINT_SUCCEEDED:
2029       return [
2030         ...state.filter(blueprint => blueprint.present.id !==
2031  action.payload.blueprint.id),
2032         {
2033           past: [],
2034  // to bridging every kind of cloud
2035           present: Object.assign({}, action.payload.blueprint, {
2036             localPendingChanges: [],
2037             workspacePendingChanges: []
2038           }),
2039           future: []
2040         }
2041       ];
2042  // And we were able to do all this because
```

Our code is open_

Red Hat

redhat.com/ourcode