November 2019

RESEARCH QUARTERLY

VOLUME 1:3

Bringing great research ideas into open source communities

Rolling your own processor: Ahmed Sanaullah builds an open source toolchain for an FPGA

Keylime: Securing the edge, one slice at a time

The Isolation of Time and Space: Partitioning Hypervisors

The post general-purpose CPU world is upon us

Red Hat

RESEARCH QUARTERLY

VOLUME 1:3

Table of Contents

From the director 02

NEWS

The post general-purpose CPU world is upon us 04

INTERVIEW

Interview with Leslie Hawthorn 07

FEATURE ARTICLES

Roll your own processor: Building an open source toolchain for an FPGA 10

Women in Tech: Plugging the Leak in the Pipeline 16

Keylime: Securing The Edge, One Slice at a Time 19

RIG Leader Perspective; UMass Lowell 23

The Isolation of Time and Space: Using a Partitioning Hypervisor to Host Virtual Machines 25

Research Project Updates 31

About the Director: 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.

FROM THE DIRECTOR

I began paying attention to computers just around the same time that Intel's x86 architecture was starting the incredible journey predicted by "Moore's Law" and made possible by the power density principle known as Dennard scaling.

Of course, the inaptly named "Law" is nothing of the sort, but the annual doubling of compute power it promised was all too real—or at least it was until fairly recently. We all learned to expect that the next chip generation would completely eclipse the current one in terms of capability and speed, and that it wouldn't be long in coming. In hindsight, what a strange thing to expect from engineering!

One of the unfortunate consequences of this predictable growth in performance was that it made systems architecture more or less uninteresting to an entire generation of engineers. What was the point of striving for incremental improvements in system performance when the next chip generation was going to come along in a year's time and render them irrelevant? Now, however, as Dennard scaling approaches its theoretical limit, things like FPGAs, custom ASICS, and other specialized accelerators are suddenly on everyone's mind, not to mention the systems architecture development, compiler

innovations, and operating system work required to enable them. In this issue, Ahmed Sanaullah's piece on the open source FPGA toolchain he has been involved in developing shows just how exciting this area is. Of course, when we have a working toolchain, we will need a more advanced way of partitioning systems to take advantage of it. Craig Einstein's thesis work on a very lightweight partitioning hypervisor, also in this issue, may provide just that.

Finding students who are interested in open source and helping them build that interest is an important part of our mission at Red Hat Research. I'm particularly happy therefore to be able to present our involvement with Tech Together, described in Sarah Coghlan's piece on women in tech. Through our sponsorship, we aim to help Tech Together, and events like it, make a real dent in gender imbalance in tech, and, particularly, in open source tech. Our discussion with community guru Leslie Hawthorn illustrates another path to the same end. By connecting open source with important real world use cases, we can attract a whole different group of people to open source than those who are just interested in tech for its own sake.

Finally, I'm really pleased to be able to share the graduation of a Red Hat Research project to a full-fledged Red Hat engineering effort. The Keylime project focuses on allowing system users to ensure for themselves that the systems they are using are running the software they claim to be. This is referred to as "attestation". Before now, the process of checking key parts of the software stack, like the BIOS and firmware, the bootloader, and the operating system kernel, was manual and tedious. With Keylime, a cloud user can attest the entire stack their code is running on, trusting only the TPM chip in the machine to correctly report cryptographic hashes of that stack's components. Keylime came out of an MIT Lincoln Labs development effort at the Mass Open Cloud, was adopted by a couple of folks in the Red Hat security group, and has now made the transition to a full product development effort. It is the kind of success story I hope will be commonplace as Red Hat Research grows, and we move more and more good ideas into open source.

Red Hat Research Quarterly delivered to your digital or physical mailbox?

Yes! Subscribe at research.redhat.com/quarterly.

Finding students who are interested in open source and helping them build that interest is an important part of our mission at Red Hat Research.

...any notable innovation on the hardware side pales by comparison to the variety of system and processor architectures of just a couple of decades ago.

THE POST GENERAL-PURPOSE CPU WORLD IS UPON US

Hardware and microarchitecture innovations for complex, data-intensive, distributed computing systems were some of the themes for the inaugural Red Hat Research Day in May. Another topic, as covered in the Red Hat Research Quarterly Volume 2, was data sovereignty in all its aspects, including privacy, governance, and security.

WHY SO LITTLE FOCUS ON HARDWARE?

For the past decade, the tech industry has mostly focused on software innovations rather than taking system hardware in new directions. To be sure, a lot of semiconductor and electrical engineering design work has gone into building out the massive scale-out x86 server farms that power the software. But any notable innovation on the hardware side pales by comparison to the variety of system and processor architectures of just a couple of decades ago.

What happened was that x86 mostly won. And, furthermore, it won in the form of largely standardized dual-processor socket rackmount servers. (Along with desktop and laptop clients.) A variety of intertwined economic and technical factors led to this outcome.

As covered by Boston University (BU) PhD candidate Han Dong in his Research Day talk, two of the most important forces that got us to where we are today are Moore's Law and Dennard scaling.

Probably everyone in the tech industry has heard of Moore's Law, the observation that the number of minimum cost components in an integrated circuit doubles about every two years. This means that you could get twice the transistors for the same cost which, historically, roughly translated into also getting about twice the performance.

Moore's Law persisted for a long time, in part because of another observation termed Dennard scaling. Dennard scaling states, more or less, that, as transistors get smaller, their power density stays constant—which also means they won't run any hotter.

One important consequence of Moore's Law and Dennard scaling was that, if you needed a faster computer, you could just wait a couple of years and buy a new one that was twice as fast for no more money and no difference in size or power consumption. Oh, and it could run the same software.

To say that this state of affairs made life difficult for any entrepreneur thinking to launch a new exotic system design is a significant understatement. Issues

of software compatibility, time-to-market, or even just perceived risk made competing with the entrenched x86 architecture challenging.

WHAT COMES NEXT?

However, both Moore's Law and Dennard scaling are petering out. The industry must now consider alternative ways of gaining performance at the hardware level, the operating system level, or even a combination of the two.

We see early examples of this trend in the widespread use of graphics processing units (GPU) and specialty processors, such as Google's tensor processing units (TPU), to accelerate machine learning and other CPU-hungry workloads.

However, there's also a great deal of work in this vein that is still in a relatively early stage of research—which is one reason why work in this area is a great fit for the Red Hat Collaboratory with Boston University (BU). BU's Orran Krieger describes how, on the one hand, research systems are often considered "toy systems" and his vision with the Collaboratory is to "create projects where we can do things together to build innovative systems that can be used directly." From Red Hat's perspective, Uli Drepper, a Red Hat Distinguished Engineer who investigates future compute architectures, sees this as "a force multiplier. It's an opportunity to work on projects which might not be [in products] in the near future."

RESEARCH DAY TOPICS

Field Programmable Gate Arrays (FPGA) was one major topic at the Research Day on the hardware side. FPGAs are semiconductor devices that connect configurable logic blocks (CLBs) via programmable interconnects. FPGAs can usually be reprogrammed for different purposes after manufacturing. FPGAs are a more flexible alternative to custom-designed Application Specific Integrated Circuits (ASICs) and, with increasing performance, are now being used for an expanding set of workloads.

As part of the Research Day talks, BU's Martin Herbordt and Red Hatter Ahmed Sanaullah presented "FPGAs Everywhere in Large Scale Computer Systems." One use case in particular that Herbordt highlighted was lossy compression, which is important in certain high performance computing applications that can generate a petabyte of data per minute. The problem is that compressing this minute of data with general purpose architectures can take half a day. With FPGAs, Herbordt pointed out that you "can do it at streaming rate."

Other talks focused on various aspects of software. For instance, Red Hat's Larry Woodman and BU PhD candidate Ali Raza discussed UniKernel Linux. Unikernels are single address space library operating systems. With a unikernal, a developer selects, from a modular stack, the minimal set of libraries which correspond to the OS constructs required for their application to run. An application compiled into a unikernel only has the required functionality of the kernel and nothing else. One longer-term goal for the use of unikernals is to provide a viable alternative to packages that, fully or in-part, bypass or avoid the kernel and do most of the processing in user space.

Other presentations covered ongoing work that is being done in areas such as latency-sensitive workloads, single-threaded performance acceleration, and novel ways to allocate memory bandwidth.

THE NEED FOR NEW TYPES OF SPEED

When he was at DARPA, Robert Colwell pointed out that from 1980 to 2010, clocks improved 3500X and micro-architectural and other improvements contributed about another 50X performance boost. The process shrink marvel expressed by Moore's Law overshadowed just about everything else. That performance knob has largely run its course, meaning that we can no longer take the broad outlines

...we can no longer take the broad outlines of today's hardware and software landscape as a given.

of today's hardware and software landscape as a given.

The bad news is that Moore's Law played a huge part in enabling today's technology landscape. Achieving a comparable pace of improvement in different ways won't be easy and may not even be possible. But the flip side is that many potential innovations that weren't of broad interest in an industry dominated by Moore's Law are now in play. There exists an enormous opportunity to research and develop these innovations, which will require different disciplines and different types of organizations to work together.

To see highlights from Research Day at Red Hat Summit 2019, go to https://www.youtube.com/watch?v=h6YOA9agi5U

Chris Wright's talk on hardware innovation https://www.youtube.com/watch?v=9sZCC73PfSo

Orran Krieger and Uli Drepper on the relationship between operating systems and hardware innovation https://www.youtube.com/watch?v=vqybu_Q_ebA

AUTHOR

– Gordon Haff, Technology evangelist, Red Hat

Orran Krieger and Uli Drepper on the relationship between operating systems and hardware innovation

THE HUMAN FACTOR IN OPEN SOURCE

Red Hat Research Quarterly had a chance to interview Leslie Hawthorn, one of the founding members of Grace Hopper Open Source Day, about motivation and experiences. Open source software had been literally all around her–all she had to do was ask and discover.

RHRQ: You followed a non-engineering path to open source, first discovering it when playing mp3 files from a GNOME desktop and then discovering the usefulness of the Mozilla Firefox browser from engineers at Google. It's fair to say that the gateway to open source software was a business productivity tool, but what was the motivation behind your lifelong engagement?

Leslie Hawthorn: I've had the opportunity to not only do work focused on open source from a business perspective, but also had the opportunity to focus on open source from more of a public good perspective, just because of the privileges that my work allowed me.

Part of the work on Google Summer of Code was to do outreach to universities, to help them understand that this was an opportunity provided by Google to help their students become competent at open source software development. Unlike typical programming experiences, it gave people all sorts of skills that were required for the job market: the ability to work well remotely, to communicate effectively in writing, to work in a distributed team etc., in addition to learning open source version control tools and resources, like Github.

At Google, I was able to act as an advisor to the Humanitarian FOSS Project, which was started by several colleges on the east coast of the U.S. The goal of the Humanitarian FOSS Project was to bring more computer science students into the world of free and open source software development, but doing so with a humanitarian focus. Significant amounts of research had shown that women, or other underrepresented groups in the technology space, were much more driven to working in STEM disciplines and, in particular in computer science, when their work was made relevant to them on a social basis. A person who doesn't necessarily see themselves as a computer scientist becomes much more interested in computer science as a discipline when they realize this gives them the opportunity, for example, to develop applications used for healthcare in the developing world, which is where their true passion lies. They want to help society and computer science becomes the vehicle to do so.

LESLIE HAWTHORN

Senior Principal Technical Program Manager, Open Source Programs Office, Office of the CTO, Red Hat

As an internationally known developer relations strategist and community management expert, Leslie Hawthorn has spent the past decade creating, cultivating, and enabling open source communities across universities, enterprises, and non-profits. She's best known for creating Google Code-in, the first global initiative to involve pre-university students in open source software development.

...while I get really excited about new software, it's less about features than what it's bringing from a social change perspective...

RHRQ: It's interesting to see how your entry into open source was through human connections. Now you're suggesting that combining tech with social good is a really good way to bring more people into the computer science fold. What do you make of that parallel?

Leslie Hawthorn: When I think about my journey to becoming a technologist, I think it's a little bit strange. I was very much a fan of the humanities and very excited about understanding the processes by which human beings communicate effectively with one another. I applied that knowledge to my work in the tech industry and how folks work together in open source software communities. And, while I get really excited about new software, it's less about features than what it's bringing from a social change perspective, like ad-blocking software for example. I think about things like smart cities, or what does the world look like when we have IoT and edge applications that are really secure, to protect consumer and business data?

I, along with some other folks who are very passionate about the idea of the importance of open source software being part of that push to have women to be more involved in technical disciplines, created something called, Grace Hopper Opensource Day. It started as a three-hour long Code-athon for Humanity in 2011, and it was a group of us just getting together to help women get started working on open source projects. Now there's an entire conference track focused on open source software.

One of the outcomes has been to get more academics talking about their work in the open source software space, and really shining a light on how their use of open source software is able to meet all kinds of requirements around data reusability, code reusability, replicability of experiments.

All of this is extremely important for projects that receive grants, e.g. from the National Science Foundation (NSF). It turns out doing things the open source way just happens to satisfy the NSF project architecture requirements. That way, you set your project up in an open and transparent way from the get-go.

I've worked with various professors on introducing open source into their curricula. For example, the Rochester Institute of Technology became the first university in the United States to offer a minor in open source software. That's a huge way to move the needle forward.

RHRQ: What are some of the barriers to the adoption of open source software and methods by universities?

Leslie Hawthorn: I think one of the places where there is sometimes a barrier to adoption of open source

software is in universities where open source software and its value is not necessarily well understood. If you look at Red Hat's research relationship with universities, Red Hat does not require assignment of intellectual property when it funds research. That is left to the schools themselves. And they are able to productize the research that comes out of those grants in whatever way they wish to, be that through the university's office of technology transfer, or companies founded, etc. But if you look at the way that most universities are incentivized to spend grant funding, if it's coming in from a private enterprise, there are hopes of developing intellectual property that the enterprise can then commercialize.

For organizations that have a very strong technology transfer office, those technology transfer offices are looking for opportunities to be able to produce a patent, or some other type of constrained intellectual property rights, that can then be monetized, and continue to drive revenue for the university after the grant funding has concluded. This is, of course, necessary since the research needs to continue even after the grant has run out.

RHRQ: Do you see these economic pressures as negatively impacting the growth of open source software?

Leslie Hawthorn: We have a generation lacking in opportunities for economic mobility. I think what we're going to see is that fewer people are motivated to participate in open source software from pure delight over the cool experiment that they're doing and more of them are going to be doing it from an economic motivation. But they still want career fulfillment and they want to do something that they think contributes to making the world a better place as a whole.

Maybe it's working on medical record systems for the developing world or maybe it's making sure that communities doing subsistence agriculture are able to get better economic value from their agricultural transactions, because they have access to open source software applications.

There are all kinds of ways in which our work can contribute to the bottom line and social good. I think that we will see more people invested in working in technology and with open source software when we return to our roots, where it's not just this code is available, and everyone can do with it as they wish, but it's also that there was a sense of social responsibility from the creator to the audience, and then the audience to everyone else as well.

–RH

ROLL YOUR OWN PROCESSOR: BUILDING AN OPEN SOURCE TOOLCHAIN FOR AN FPGA

Field Programmable Gate Arrays (FPGAs) are rapidly becoming first-class citizens in the datacenter, instead of niche components. This is because FPGAs **i)** can achieve high throughput and low latency by implementing a specialized architecture that eliminates a number of bottlenecks and overheads of general purpose computing, **ii)** consume little power and, by extension, have a high power-performance ratio, **iii)** have high-speed interconnects and can tightly couple computation with communication to mask the latency of data movement, and **iv)** can be configured so that each design is tuned for individual use cases.

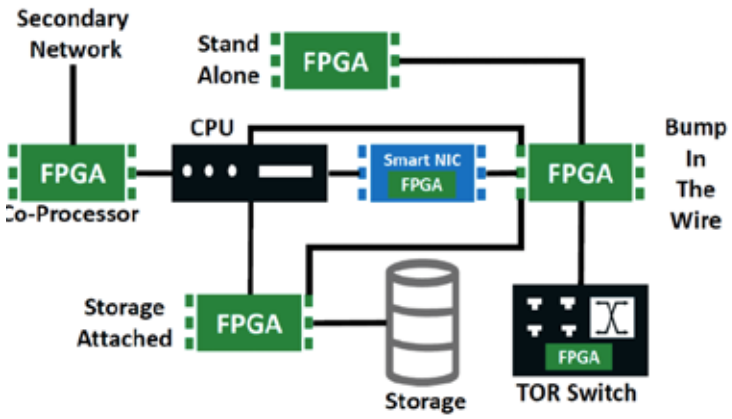


Figure 1: *Different configurations for deploying FPGAs in Data Centers*

Figure 1 illustrates the different configurations in which FPGAs are being deployed in a datacenter. Bump-in-the-Wire (BitW) FPGAs process all traffic between a server and a switch to perform application and system function acceleration. Coprocessor FPGAs provide a traditional accelerator configuration, like GPUs, with an optional back-end secondary network for direct connectivity between accelerators. Storage-attached FPGAs process data locally on storage servers to avoid memory copies to compute servers. Stand-alone FPGAs provide a pool of reconfigurable accelerators that can be programmed and interfaced with directly over the network. Smart network interface controllers (NICs) contain embedded FPGAs which perform custom packet processing alongside a NIC ASIC (application-specific integrated circuit). Finally, network switches can also contain embedded FPGAs that process data as the data moves through the datacenter network (e.g., collective operations such as broadcast and all-reduce).

RESEARCH PROBLEM

FPGAs have traditionally lacked the clean, coherent, compatible, and consistent support for code generation and deployment generation that is typically available for traditional central processing units (CPUs). For the most part, previous efforts to address this have been ad hoc and limited in scope. Those who try to address this always do something special due to poor tooling, and the tooling that does exist is insufficient, especially for datacenter and high-performance computing (HPC) applications.

This is because of the heavy reliance on proprietary, vendor-specific tools for core operations. These tools can change frequently and

significantly, which means that even if we wanted to not be ad hoc, for the most part we couldn't be without investing a lot of work which could be wiped out with a new generation of FPGAs. Moreover, these tools are not necessarily aimed at providing the most efficient solution since they limit the "flexibility" offered to users. For example, these tools **i)** do not allow modifications the algorithms for core operations (such as logic optimization and place & route), **ii)** hide details of (and access to) the underlying device architecture which prevents implementation of important functions (such as logic relocation without recompilation), and **iii)** are designed to be generic with limited opportunities for customization (such as vendor IP blocks).

This is not a problem unique to FPGAs, however. Similar issues already exist for software. Therefore, similar to free software in the software world, we must be able to code and deploy custom architectures using transparent, open, end-to-end frameworks that are **i)** not tied to any vendor, that is, do not use IP blocks or tools that are only compatible with the FPGA boards of a particular vendor, **ii)** provide opportunities for customization across all levels of the development stack, and **iii)** can be upstreamed, that is, can be easily and reliably integrated into downstream projects in order to build more complex and intricate systems.

HARDWARE AS A RECONFIGURABLE, ELASTIC, AND SPECIALIZED SERVICE

We refer to our framework for providing upstream support for datacenter FPGAs as "Hardware as a RecoNfigurable, Elastic and Specialized Service" (**HaaRNESS**). HaaRNESS is built as a high-level synthesis (HLS) tool, which creates and deploys high-quality hardware from algorithms expressed in high-level languages (HLL) such as OpenMP or OpenCL. Developers only specify the algorithm, with minimal use of pragmas and low-level constructs, and hence require virtually no prior expertise in hardware development; this prior expertise is both in terms of hardware-specific languages (e.g HDL), as well as code structures (in HDL and HLL) needed to effectively map design patterns to hardware. A **preprocessor** transforms this simple HLL code into an FPGA-centric HLL code (HLL*) which removes hardware optimization blockers and helps infer opportunities for parallelism. Then, an **HLS compiler** converts this HLL* code into HDL (hardware assembly language). The resulting HDL is run through a **system generator** which can perform one of two operations**: i)** cycle accurate simulation, or **ii)** deployment of application logic on the physical FPGA system. In case of the latter, a **bitstream compiler** maps the HDL code onto the FPGA fabric using Synthesis and Place & Route. Then, a **software runtime** is used to program the application onto the board and interface with it. Finally, similar to the OS on CPUs, a **hardware operating system** (OS) is provisioned on the FPGA in order to share the FPGA fabric amongst multiple independent entities.

HLS CODE PREPROCESSOR

Current HLS tools can require developers to explicitly identify opportunities (and constraints) for parallelism, as well as manually implement a number of important design features such as caches, loop coalescing, function inlining, floating point accumultors and data hazard elimination. This substantially increases the complexity of HLS code that developers need to provide. Our HLS code preprocessor reduces this complexity by automatically identifying optimization blockers in an HLS compiler through compiler instrumentation, and then addressing them using a series of system code transformations. Optimization blockers occur when a compiler writer is not being allowed to infer an optimization. An optimizing transform may be blocked if

it:
i) modifies code functionality, instead of structure only, **ii)** can result in a failure to compile, **iii)** is based on information available at run-time, **iv)** requires a global view of the computation, and/or **v)** is based on implicit code behavior that may be visible to the developer, but cannot be reliably extracted by the compiler.

Figure 2 illustrates our approach. To identify optimization blockers, we first built a logical model for FPGAs by identifying a set of core design patterns that an HLS compiler should be able to infer and implement efficiently in order to achieve high quality code generation. Examples of these design patterns include single instruction, multiple data (SIMD), pipelining, caching, logic inlining, and loop structures.

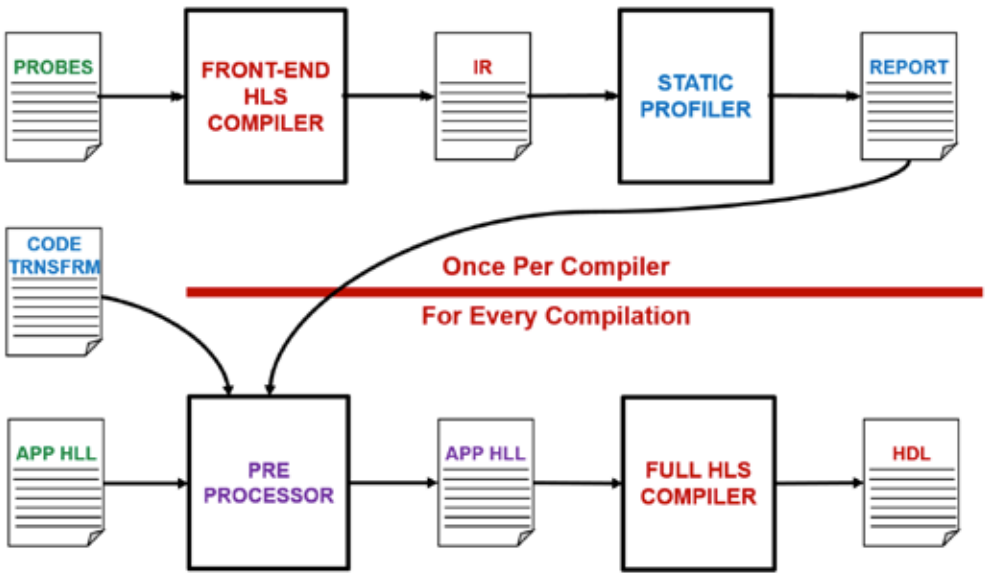


Figure 2: *Framework for automatically removing optimization blockers using compiler instrumentation*

Then, we instrumented the HLS compiler (OpenCL in our proof-of-concept) to determine what it has inferred given an input code. This requires analyzing the IR at compile time (static profiler) after all optimizer passes have been run (i.e. output of the front-end HLS compiler).

We then built a set of probes which contain individual design patterns in relative isolation, so that we can determine compiler effectiveness for each. By running these probes through the compiler and looking at instrumentation report, we can tell what optimizations are blocked. The process is done once for a given version of an HLS compiler. Along with a set of probes, we also provide a set of HLL-HLL code transforms that can remove the optimization blocker for each probe. Examples of these transforms include loop unrolling for SIMD and generating register caches for read-after-write hazards for on-chip memories.

These transforms are only done if an optimization is blocked. Finally, this set of code transforms and the probe report is fed into the pre-processor.

ADVANCING HLS COMPILERS

Current HLS compilers have two major drawbacks. First, since existing HLS tools map code fragments to vendor IP blocks in order to generate HDL from HLL code, a large library of such blocks is typically needed. Such libraries consume a large amount of CPU memory, have high-overhead non-trivial lookup operations, and provide limited opportunities for optimization since they are proprietary. Only a limited set of parameters can be modified and that, too, is within predefined bounds.

Second, it is also likely that a significant fraction of code that is translated to these IP blocks is not needed for the HDL to execute. Application logic can have a spectrum of performance requirements for different components, not all of which require execution on custom hardware, e.g. control plane

versus data plane.

Our goal is to advance HLS compilers by addressing the above two drawbacks. The first enhancement is to reduce the size of code sequences being translated to hardware, and perform this translation using only basic vendor-agnostic and transparent hardware building blocks like registers and gates. This enables faster compilation times and allows the design to be tuned for each individual application.

The second, perhaps more critical, improvement is to identify, at compile time, the best approach for implementing the algorithm. For the code generation itself, we have three different pieces: **i)** the part which must always executed on the host and cannot be on the FPGA e.g. due to I/O, **ii)** the part which is translated into softcores on the FPGA, and **iii)** the rest of the code which is translated into HDL. These parts can be either inferred automatically by the compiler (directly or through profiling executables), or marked up using OpenMP primitives. The split of **(ii)** and **(iii)**, in particular, is important, since functions implemented using softcores consume negligible resources (logic/memory/DSP blocks) and can achieve asynchronous operation with respect to HDL. Moreover, for part **(ii)**, we eliminate Place & Route (an hours/days long process), and achieve CPU-only software-like turnaround times, because the HDL does not change. If the HDL itself is as small as possible for computation kernels and/or is relocatable to other parts of the FPGA fabric, the need to run Place & Route is further reduced.

SYSTEM GENERATOR: CYCLE ACCURATE SIMULATION

Another major component of our research includes building a cycle accurate simulation framework. The framework can estimate performance directly from HDL code without compiling to actual hardware, because rapid and reliable design space exploration substantially reduces turnaround times for building high quality hardware. While this feature, called RTL simulation, is certainly not novel, we provide significantly more control over what can be evaluated and how.

Using our framework, developers have the flexibility of testing both the application logic and its interaction with the world around it. The latter involves testing application logic after connecting it to intra-FPGA wrappers, operating systems, external devices, etc. This is important since testing the application logic only, without modelling the environment around it, can result in developers converging on a design that gives worse performance than naive code when actually implemented on an FPGA. Worse, a design is likely to fail to execute altogether if deadlocks were not

The second, perhaps more critical, improvement is to identify, at compile time, the best approach for implementing the algorithm.

Using our framework, developers have the flexibility of testing both the application logic and its interaction with the world around it.

properly identified beforehand.

SYSTEM GENERATOR: DEPLOYMENT

Bitstream compiler

The mapping of HDL to the FPGA fabric is typically done exclusively by FPGA vendors. This is because it requires knowledge of low-level details of the underlying FPGA hardware, which vendors typically do not disclose publicly in order to protect intellectual property. Lack of these low-level hardware details means that we cannot determine how designs map to FPGAs and thus guarantees of security and performance cannot be reliably provided.

Our research focuses on inferring low-level hardware by reverse engineering FPGA bitstreams. The goal here is to obtain key insights into the compilation processes, which we can then use to build an open bitstream compiler. This allows us to both reduce the limitations of proprietary bitstream compilers, as well as implement important features that are currently not supported, e.g. FPGA fabric attestation.

Software runtime

With regards to software runtime, our research is primarily focused on building vendor agnostic tools, such as drivers and runtime libraries. Similar to how the Linux kernel is built, our goal is to separate the software stack for FPGA tools into architecture/configuration dependent and independent components. This will enable us to maintain a uniform and reusable structure for software runtime across all types of FPGA configurations boards in the datacenter. It will also reduce the complexity of adding and removing features, since well-defined APIs will ensure that changes are compatible with existing code. Having these APIs map well to a broad set of vendors is possible since FPGAs talk to host machines over standard buses, such as PCIe (peripheral component interconnect express). These standard buses, and associated protocols, constrain the behaviour of both the software (drivers) and hardware (PCIe logic blocks) built around the buses in a similar manner for all FPGAs. Any subtle differences, such as vendor or device IDs, can be supplied as compile/load/run-time values. As a result, it is possible to implement reliable and effective uniformity, at least for the lower levels of the hardware and software stacks, and expose a consistent interface to applications on the host and device.

Hardware OS

Hardware operating systems are effectively any logic on the FPGA that is not part of the application. They are responsible for partitioning the device fabric between multiple entities, data flow management and

interfaces, and hardware modifications. They also manage the flow of data between different components in the FPGAs by defining a number of specifications such as APIs, protocols, bus widths, clock domains, FIFO depths, etc. **Figure 3** shows our hardware operating system for Bump-in-the-Wire FPGAs, called Morpheus. Morpheus supports the sharing of the FPGA fabric between developers and the system administrator. Administrator functionality offloads are particularly useful for accelerating a large number of critical workloads such as encryption, SDN, Key Value Store, database operations, etc.

Since APIs are well defined, Morpheus can be easily modified to support other deployment configurations of FPGAs in the datacenter. This is critical to ensuring compatibility across the stack, enabling portability across FPGAs, and reducing developer effort in integrating their designs into the hardware OS.

CONCLUDING REMARK

We expect to have a toolchain, Morpheus, and compiler extensions to target the FPGA in the near future. If you are interested in collaborating, please feel free to contact us and we would be happy to discuss the research with you: asanaull@redhat.com.

AUTHOR

– Ahmed Sanaullah,PhD
Software engineer, Office of the CTO

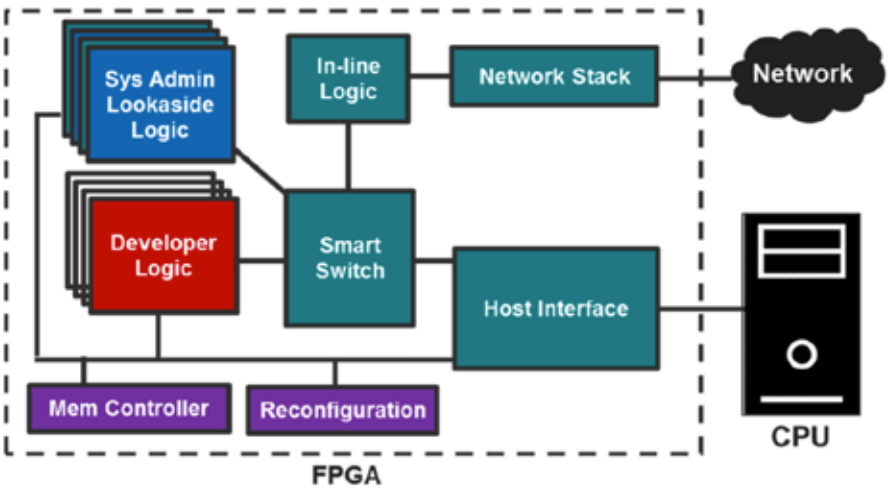


Figure 3: *Design of our prototype Hardware OS for Bump-in-the-Wire FPGAs*

AHMED SANAULLAH is a software engineer for the Red Hat Office of the CTO, working on all things FPGA. This includes a number of projects across the hardware and software stacks, such as compilers, drivers and hardware operating systems. He has a PhD in Computer Engineering from Boston University, a MSc in Electrical and Electronic Engineering from The University of Nottingham, and a BS in Electrical Engineering from Lahore University of Management Sciences.

WOMEN IN TECH: PLUGGING A LEAK IN THE PIPELINE

Source: *United States National Center for Women & Information Technology*

It's hardly news that there is an enduring gender gap in tech. According to the United States National Center for Women & Information Technology, only 26 percent of the U.S. computing workforce is female. Women are chronically underrepresented in the U.S. tech sector and this lack of diversity is not a recent phenomenon.

Although the problem is complex and finding solutions is daunting, it's entirely possible to bring more girls into tech and support them at all points along the pipeline. Women-led organizations like Girls Who Code and littleBits were founded with this mission in mind: to get more girls interested in technology in the first place, and then support them throughout their journey so they stay interested, persist, and succeed. Industry can play a powerful role in supporting women even further as they enter the workforce.

This year, Red Hat is a top-tier sponsor for TechTogether Boston, Boston's largest all-female, femme, and non-binary hackathon, to be hosted at Boston University (BU). TechTogether Boston's mission is to help gender-marginalized groups thrive and be more successful, confident, and prepared for careers in the tech field. Red Hat attended last year and hired 10 strong interns from the program. This year, we're aiming to double the number of hires drawn from the talent pool participating in this event.

"In high school and college, women fall out of the tech pipeline because there aren't the resources there to support them in their tech-related classes, clubs, internships, and even hackathons," says Grace Yeung, Director of Marketing at TechTogether Boston. "By being a top sponsor for TechTogether Boston, Red Hat is directly helping to plug a leak in the tech pipeline."

In 2018, only 20% of hackathon participants identified as women. TechTogether Boston is directly addressing this by creating an environment for traditionally gender-marginalized people who aspire who aspire to work on projects, explore their interests on a deeper level, and connect with other women and non-binary people in technical fields and build lifelong connections.

"With the support of sponsors like Red Hat, we are able to host our event free of charge to all our hackers, thereby eliminating the financial barrier that is present in trying to break into tech," Yeung says. "With their on-the-ground presence at our event, Red Hat allows women and non-binary individuals to see that technology companies are willing to invest in them and hire them as well."

Red Hat is planning to provide the best experience for these hackers by offering participants the opportunity to network, attend a variety of workshops, interview for internships, enjoy fun activities, and make lasting friendships along the way. Tech Together not only supports the development of new talent, but also celebrates strong female role models in the local technology industry. Last year, Red Hatters Oindrilla Chatterjee and Hema Veeradhi each led workshops at the event. Data scientist Chaterjee presented "Understanding Text and the Underlying Sentiment," while software engineer Veeradhi covered "Machine Learning Flow on OpenShift." Chatterjee and Veeradhi are both recent graduates of Boston University.

Efforts to reduce the gender gap reach beyond Boston. This past summer, Red Hat continued its sponsorship for the University of Massachusetts (UMass) Lowell's RAMP program. RAMP, which stands for Research, Academics and Mentoring Pathways, is a six-week program for twenty first-year female engineering students. RAMP is designed by the faculty based on previous experiences mentoring female students. The guiding principle behind this program is that when women entering engineering don't stay the course, other young women then feel isolated and switch to other majors.

"Often times, subjects like math and science breed unwelcoming environments from day one," Kate Carcia, associate manager in quality engineering at Red Hat and graduate of UMass Lowell, says. "We rob people of the opportunity to learn if we shut them out from the beginning."

Without other women to look up to, many young women self-select out of a technical career path before they have even given it a chance. Leaning on her own experiences, Carcia mentored students from the RAMP program and spoke on

SARAH COGHLAN *is the University Program Manager for Red Hat Research working out of the Boston Office. She oversees the PnT Intern and Co-op Programs and is the lead organizer for Tech Together Boston.*

the program's industry panel with other female engineers from Red Hat.

"I would not have stuck around if it weren't for my mentors," Carcia says. "The feeling of not belonging was enough to start pushing me out the door on numerous occasions. I'm lucky to have people who push me right back in."

There is a huge opportunity to shift the trajectory of women and girls entering the industry and make tech an exciting and welcoming career opportunity for all. Changing how people envision computer scientists is an important step in the effort to encourage more women to pursue careers in technology. Girls need to see that computer scientists come in all shapes and all sizes. Adding more women to Red Hat will attract more women to Red Hat - and we are doing just that.

AUTHOR

– Sarah Coghlan

ARTICLE LINKS:
26 percent of the U.S. computing workforce is female: *http://www.techrepublic.com/article/the-state-of-women-in-technology-15-data-points-you-should-know/*

Girls Who Code: *http://www.girlswhocode.com*

littleBits: *https://littlebits.com/*

TechTogether Boston: *https://boston.techtogether.io/*

20% of hackathon participants identified as women: *http://ladyproblemshackathon.com/our-impact/*

Red Hat Research Quarterly delivered to your digital or physical mailbox?

Yes! Subscribe at research.redhat.com/quarterly.

KEYLIME: SECURING THE EDGE, ONE SLICE AT A TIME

As the number of workloads in cloud, IoT, and edge continue to grow, how do we verify that a remote, shared, and/or physically unsecured computing system has not been tampered with? Is there a good way to use a standardized cryptographic module to establish a hardware root of trust, do a remote, trusted secure-measured boot, do remote integrity verification management, and do remote encrypted payload execution?

Keylime is an open source community-based project that enables the establishment and maintenance of trusted compute in distributed deployments. It uses embedded Trusted Platform Module (TPM) hardware (version 2 and later) and the Linux kernel's Integrity Measurement Architecture (IMA) subsystem to do so. Keylime was originally created by an MIT Lincoln Laboratory research team. It has now grown to include a small and dedicated open source community behind it.

WHAT PROBLEMS DOES KEYLIME HELP SOLVE?

Today's clouds rely upon complete trust in the provider to secure applications and data. Cloud providers do not offer the ability to create hardware-rooted cryptographic identities for cloud resources nor do they supply sufficient information to verify the integrity of systems. Trusted computing protocols, as well as trusted hardware like TPM chips, promised a solution to this problem. Unfortunately their complex implementation, low performance, and lack of compatibility with virtualized environments has limited their adoption.

Keylime's design allows the remote attestation and IMA monitoring of thousands of nodes. Work is underway in collaboration with Boston University (BU) on a virtual TPM (vTPM) quote. Keylime can also scale to be used to monitor thousands of virtual machines running on a single host by reducing the performance penalty of directly calling the hardware TPM of the cloud node to cryptographically sign data.

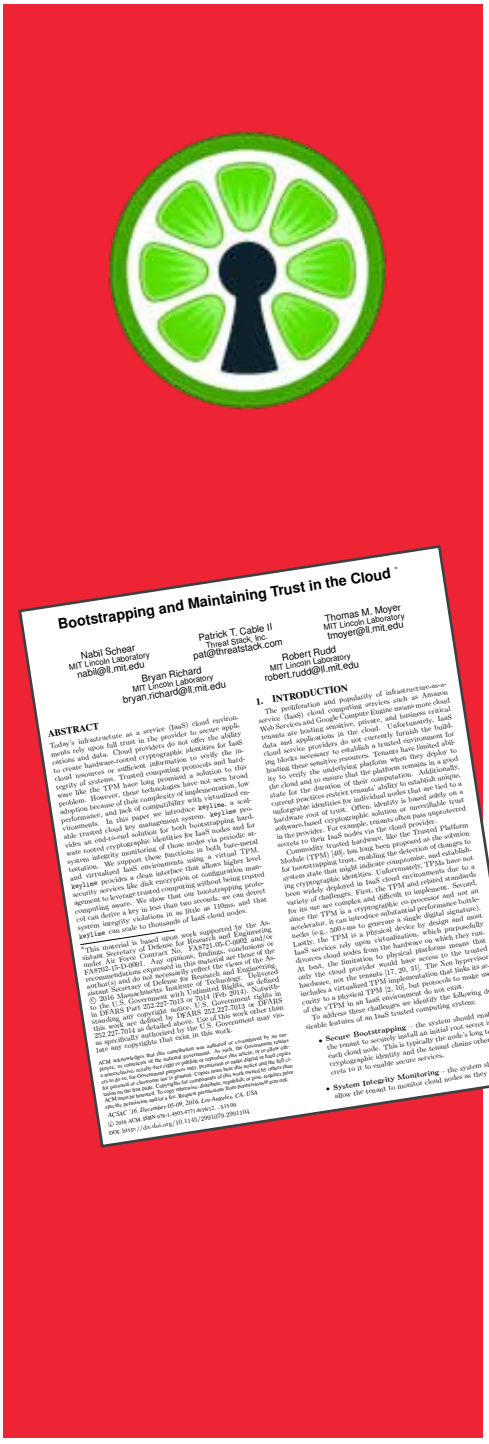
KEYLIME'S BENEFITS

Keylime enables:

1. Trusted measured boot functionality and secrets provisioning using encrypted payloads.
2. Runtime integrity checks and verification.

It also supports the following distribution scenarios:

- Single site - single node (multi-user)



The Keylime community is currently working on packaging the project for different platforms and hardening the system by porting subsystems from Python to Rust.

- Single site - multi-node (Datacenter, IoT)
- Multi-site - multi-node (Distributed Datacenter, Network Edge equipment, IoT)
- Multi-tenant (Cloud)
- Baremetal
- Virtual machines (VM)

Figure 1 and **Figure 2** illustrate the difference between a traditional remote trusted secure boot and continuous remote integrity verification.

THE TPM CORE

The TPM chip provides a root of trust facility introduced by the Trusted Computing Group (TCG), standardized in 2009, and updated in 2015 to version 2.0. The TPM standard includes general system trust facilities such as random number generation, secure key generation, data encryption, and remote attestation. Version 2.0 is not backward compatible with previous TPM versions. Keylime targets version 2.0 or later of the standard.

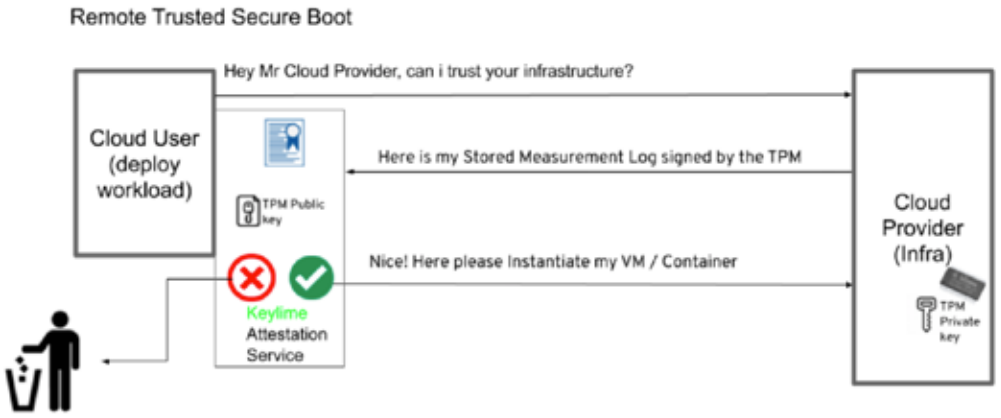


Figure 1

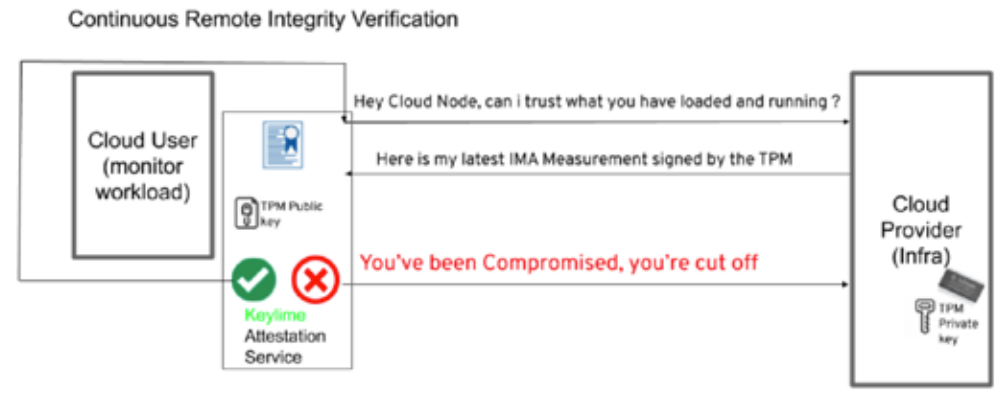


Figure 2

KEYLIME'S COMPONENTS

Using the below components, Keylime attests system integrity during node provisioning as part of a trusted boot workflow as well as continuously attesting the trustability of the runtime environment while it is operational. The only external component Keylime needs to operate is a functional TPM provided by each infrastructure node where attestation is desired.

The three main components to the Keylime system are the **agent**, the **registrar**, and the **verifier**. All three components were initially developed in Python. Components that have greater performance and security needs, like the agent, are being ported to the Rust language for its performant nature as a low-level systems language and for the strict security model of ownership enforced by the compiler.

The agent is required to be installed on each node in the infrastructure where attestation is desired. It is responsible for interacting with the TPM of the system it resides on, including TPM 2.0 functions such as requesting cryptographic quotes. The agent is then responsible for communicating the collected information back to other system components to enable the processing of the trust chain.

The verifier is responsible for bootstrapping a new node into the system and continuously requesting the quotes from each agent component in the system. The verifier then performs the attestation on the quotes returned to determine if there have been any unauthorized changes to the remote systems.

The registrar is responsible for maintaining the set of known secure (public) key values used during attestation processing. The agent on each node registers itself with the registrar upon boot up, locking in the initial state of the node for later comparison. The registrar's secure key set also includes the public keys for the hardware manufacturer of each node in the system. These manufacturer keys are used to verify that the hardware TPM is valid and can be used as the root of trust for the respective node.

Also included in the Keylime tooling is a tenant command line interface (CLI) utility (keylime_tenant). The tenant utility uses Keylime's RESTful interfaces to communicate with the Keylime components. The user can either employ the tenant utility, the Keylime web user interface (UI), or integrate a management system with Keylime by integrating with the Keylime REST application programming interface (API) directly.

LUKE HINDS *is a software engineer in Red Hat's CTO office with a focus on security trust systems for Cloud, Edge and IoT.*

ANDREW TOTH *is a software Engineering Manager in Red Hat's CTO office focused on Telecommunication Service Provider solutions.*

Keylime includes a simple Certificate Authority (CA) manageable by the tenant utility or its dedicated Keylime CA utility (keylime_ca). The CA is an integral part of initially establishing trust during the bootstrapping phase of node provisioning and in enforcing the trust relationship of the node thereafter. The CA is initially responsible for signing all boot keys sent to the nodes being provisioned, establishing the initial trust the system relies on. If the verifier detects a breach of the established trust via broken attestations, the CA is notified and expected to revoke the trust by invalidating the keys associated with the compromised node.

WHAT'S NEXT FOR KEYLIME

The Keylime community is currently working on packaging the project for different platforms and hardening the system by porting subsystems from Python to Rust. Rust will provide performance improvements and brings extra security benefits due to its type ownership model enforced by the compiler

To learn more about Keylime

Interested in learning more or trying Keylime out for yourself? Come check out the community at https://keylime.dev and explore the "Get Started With Keylime" guide.

AUTHORS

– Luke Hinds and Andrew Toth

RIG LEADER'S PERSPECTIVE: ONBOARDING A NEW UNIVERSITY

A Red Hat Research Interest Group (RIG) is a self-organizing team that drives support for and participation in local technical research projects at academic and government agencies or industrial groups.

Red Hat Research Interest Groups are, in part, driven by the passion of their members. Often, these passions revolve around technology areas or social concerns and, sometimes, a combination of both. But, even with passion, pulling together partnerships with local universities is not without its challenges.

The MetroWest RIG covers Boston, Massachusetts, a city with a high concentration of universities, as well as Boston's suburbs, including the New England technology belt where Red Hat's second largest engineering office is located in Westford. While the Westford office has many research projects and internships to offer, it has fewer candidates in the immediate area. We did not have to look too far away, however, to find excellent prospects in the nearby University of Massachusetts at Lowell (UMass Lowell), where a lot of us studied and where Red Hat software engineer Jeff Brown served as an adjunct professor.

First problem solved--we had insight and connections. We also believed we had something of value to offer to the university and to the students themselves. But how could we motivate them to collaborate with us? We hoped that, although Red Hat is not as big a fish as Google or Amazon, to the people at UMass Lowell, we would still be seen as a medium-sized fish. In the end, what we found the real motivator to be was the kind of partnership we put together for them.

The next challenge was more difficult to address—finding the right channels to formalize the activities we planned to do in regards to Women in Engineering and Women in Science seminars and teaching classes as a partnership with commitment from both sides. Our entry point was the UMass Lowell Co-op Program office. Through their Professional Co-op program, students can now receive university credits and earn a salary for their work experience at Red Hat. Students who participate spend 6 months embedded in an engineering team with the ability to make significant contributions to open source projects. We now have 20 openings for UMass Lowell Co-op students.

After determining the channel to connect with the students, we were

One of the projects we sponsored was the University of Massachusetts Lowell Cyber Range, which emphasized the importance of security in open source development. The cyber range supports research, teaching, and workforce development, providing a live, sandboxed environment where students can build and investigate security systems from the ground up, safely exploring security attacks and defense techniques.

...there's a "vibe" that's reminiscent of the early days of Red Hat.

ready to tackle the final challenge—building up collaborative research projects. We talked to the heads of the computer science and electrical and computer engineering departments about the kinds of opportunities we could offer. Ideas included a semester's project, a senior project for undergrads, a master thesis, and even a multi-year PhD level project. Red Hatters came up with the ideas for the projects and dedicated at least half a day each week to mentor students on campus or at the Westford office.

Although we offer some stipends for the larger projects, the relationship between UMass Lowell and Red Hat is more about the exchange of knowledge and creation of opportunities for collaboration. The relationship is mutually beneficial. For example, the open source Linux project upstream from Red Hat Enterprise Linux, Fedora, is being experimented with and tweaked for a range of small IoT devices and facial recognition technologies by students and, at the same time, they're working on something they're calling "Fedora for Academia."

The openness of Red Hat's approach is very appealing for the university. When they have a huge program with a conventional technology company, it is difficult to get papers out or advertise the research. Someone working on a radar system for a private company might find it impossible to publish a paper. But with Red Hat, it's not a problem. We are open and all the work is open, so we don't mind when they publish their work.

UMass Lowell has a special appeal to Red Hat, as well. Besides the expertise in advanced networking and 5G that their students bring, there's a "vibe" that's reminiscent of the early days of Red Hat. As a public university located in a historical industrial center, it is often overlooked and has to work hard to be noticed or make an impact. Nobody believed in Red Hat at first, or in its commitment to open source software, and we had to work hard to prove ourselves. That's a big reason why we feel that the UMass Lowell and Red Hat partnership is going to thrive.

AUTHOR

– Rashid Kahn, Sr. Director of software engineering, Red Hat

RESEARCH
QUARTERLY

VOLUME 1:3

THE ISOLATION OF TIME AND SPACE: USING A PARTITIONING HYPERVISOR TO HOST VIRTUAL MACHINES

For certain classes of systems and applications, the latency, nondeterminism, and resource sharing found in traditional hypervisor-based virtual systems are unacceptable. Examples of this can be seen in machine control, applications guaranteeing a strict quality of service, real-time systems, and power-constrained platforms. Each of these examples would encounter dynamics in the traditional hypervisor approach that could cause the system or application to not operate as intended. For these cases, making use of a virtualized system may still be desirable if different components of a platform, or different applications run on that platform, need to make use of different operating systems. However, it might be inefficient, or too power intensive, to use a different computational platform for each operating system. Thus, another approach to virtual systems must be used to mitigate these unacceptable dynamics. One such approach is the use of a ***partitioning hypervisor****.*

Virtualization is a powerful tool that enables the hosting of multiple operating systems, known as virtual machines (VMs) or guests, on the same physical platform. This allows for the functionality and services of several, potentially different, operating systems to coexist and make use of the same set of hardware resources. These hardware resources include processors, memory, and I/O devices such as network cards, USB ports, and cameras. The use cases of such a system configuration are plentiful, spanning across several domains such a servers, robotics, vehicles, general development, and many more. Virtualization provides interesting dynamics that can exploit the hardware features of a physical platform.

Depending on how virtualization is implemented though, this exploitation has a limit. Traditionally, virtual systems make use of a hypervisor to manage the multiple operating systems' usage of hardware. This hypervisor, otherwise known as a virtual machine manager, or VMM, is a sort of resource manager. It ensures that each guest on a platform can make use of the hardware that it should be allowed to use, that each guest gets a turn to use shared resources, and that data is returned to the correct guest. A diagram of an example system running three VMs that makes use of a hypervisor can be seen in **Figure 1**.

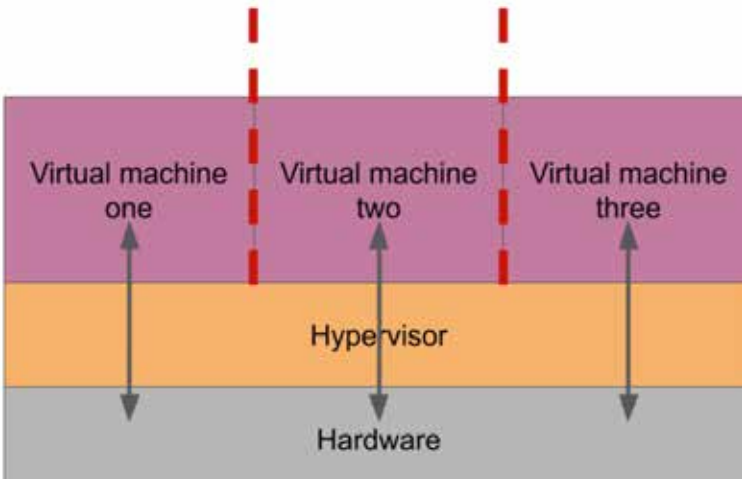


Figure 1. *This figure shows an example of a virtual system making use of a traditional hypervisor. The regions dedicated to each guest are delineated by the dashed red lines.*

In certain cases, a guest can be made aware of the hypervisor and that it is running in a virtual

environment, but in other cases a guest operating system will remain oblivious to the hypervisor.

In the first case, a mechanism known as paravirtualization is used to make the guests aware of the hypervisors existence. Paravirtualization, in essence, is when an operating system communicates with the hypervisor directly to access the hardware resources of the platform on which they are running. A guest's device drivers are altered to make calls into the hypervisor to perform tasks on its behalf, instead of the guest trying to directly access devices.

In the second case, when each guest is unaware of the hypervisor, a guest attempts to make direct use of the hardware. When this happens, the instructions that the guest attempts to execute are trapped (redirected) into the hypervisor. The hypervisor then emulates the guest's usage of a device and returns any data back to the appropriate guest. While these two dynamics are useful in allowing each guest to make use of the hardware devices of the platform, uses of the hypervisor are complicated, expensive, and nondeterministic.

When an operating system running in a virtual environment makes a request to use a device, it is unaware that other operating systems might also be making use of that device. Thus, the OS might have to wait an unexpected amount of time in order to make use of that device and to receive the expected data. Additionally, each round-trip into the hypervisor (performing a VM exit and a VM enter) can cause a TLB flush.

The TLB (translation look-aside buffer) is used to store address translations, so that a guest operating system does not always need to re-translate virtual addresses into a physical address. Once populated, the TLB reduces memory access times and improves the performance of the system. Flushing the TLB negates this performance boost and thus with every call into the hypervisor guest performance suffers. Aside from the nondeterminism and high round-trip time, hypervisors need to employ additional control structures to manage multiple guests' usage of the hardware.

All of these factors have an impact on both the spatial and temporal performance of the virtual systems. On modern hardware, these effects might not be immediately noticeable; modern platforms have suitable processing speeds and memory to handle multiple guest requests without significantly degrading performance (besides in extreme cases).

However, these systems cannot guarantee dynamics such as hard real-time performance. In this context, real-time means that a task or event will finish within a predetermined deadline and can happen periodically, with each occurrence finishing within the expected time-bound. Because of the non-deterministic nature of the hypervisor, and because of the competition for shared resources, a typical guest operating system cannot

safely make the claim that a certain task will finish in a specifically predetermined amount of time.

Hypervisors also have limited device power management. Because the hypervisor manages the hardware resources for a set of guest operating systems, if a certain device is exposed to multiple guest operating systems, any one of those operating systems could make use of the device at essentially any time. Thus, the hypervisor cannot safely give a guest OS the capability to power off the device as another guest might need to make use of it (or could be in the middle of using it!). The hypervisor is given the ability to power down and idle devices, but it can not be certain that one of the guests will not need to make use of it in the near future.

What, then, can be done to preserve the advantages of virtualization while still obtaining the guarantees a real-time system needs to operate correctly? A partitioning hypervisor is one answer: it can create a virtual system that mitigates the described latency, nondeterminism, and shared resource dynamics found in traditional hypervisors, while preserving the flexibility and low friction of VMs. A partitioning hypervisor is a form of virtual system which partitions the hardware resources of a physical platform. It then individually assigns these partitioned resources to a set of guest operating systems running in the virtual system. Each guest can then make exclusive use of the set of resources to which it is assigned. Using this method, each guest receives a dedicated set of processors, a dedicated region of memory, and a dedicated set of I/O devices. This partitioning arrangement enables each guest operating system to be both temporally and spatially isolated from each other. A diagram of an example system running three VMs that makes use of a partitioning hypervisor can be seen in **Figure 2**.

Each guest operating system in a partitioning-hypervisor-based virtual system is set up by a monitor. During this set-up, the monitor establishes the guest's memory region, ensures that the guest only has access to the set of devices to which it is assigned (processors and I/O devices), and can handle certain faults of the guest. After this set-up, the monitor largely removes itself from the guest's runtime operations. This prevents costly VM exits and enters and allows each guest to make direct use of the hardware assigned to it without any hypervisor intervention.

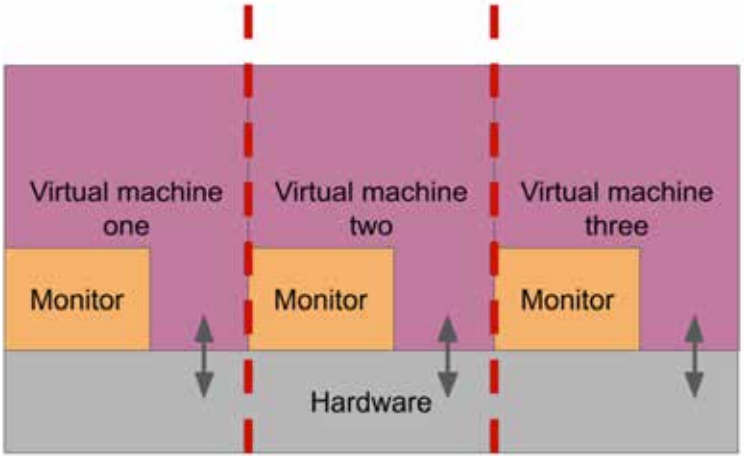


Figure 2. *This figure shows an example of a virtual system making use of a partitioning hypervisor. The regions dedicated to each guest are delineated by the dashed red lines.*

The monitor can also be used to establish shared memory communication channels. These channels

...the isolation that partitioning hypervisors provides is useful in a server and data-center context.

allow guest operating systems to communicate directly with each other through shared memory, instead of using some I/O device as if they were running on separate physical platforms, which is the traditional approach. This allows for information generated in one guest to be quickly disseminated to, and consumed by, another guest operating system. This shared memory communication enables interesting inter-guest dynamics to be established.

As each guest can interact with the hardware directly and is the exclusive user of that hardware, it has the ability to manage the power of the hardware to which it is assigned. This allows for per-guest power management of devices. This per-guest management of power allows a guest to suspend its devices in periods of low activity, thus consuming less power per-guest. In this way, less power is consumed by the platform as a whole.

The isolation between guests allows for systems of different criticalities to be run on the same platform. Criticality indicates a system's sensitivity to a certain domain, such as safety, timing, or security. Thus, a system with a high-timing criticality might be running tasks that are very sensitive to variations in time, or a system with a high-security criticality might store sensitive information that should be shielded from external readers.

Certain guests might have a high sensitivity to multiple domains, with an example of such a system being an autonomous vehicle. A vehicle has stringent timing dynamics, as the delay in the sending of a command to a wheel or propeller could cause disastrous consequences. Security-wise, it would be undesirable for an external attacker to gain control over the vehicle as unwanted commands could then be sent and processed by the vehicle. Safety-wise, it is important for the vehicle to ensure that it is fault-tolerant and will not perform any unwanted behaviour that might harm the individuals within or outside the vehicle. Using a partitioning hypervisor, this vehicle might host a guest to provide the necessary safety, security, and timing required to control the vehicle, and also host a separate, isolated guest to communicate with external sources or to perform autonomous computation that should not interfere with, but inform, the critical vehicle control.

Aside from vehicles and other IoT devices, the isolation that partitioning hypervisors provides is useful in a server and data-center context. Allowing each guest to manage its own resources enables each one to optimize its utilization of the hardware resources to

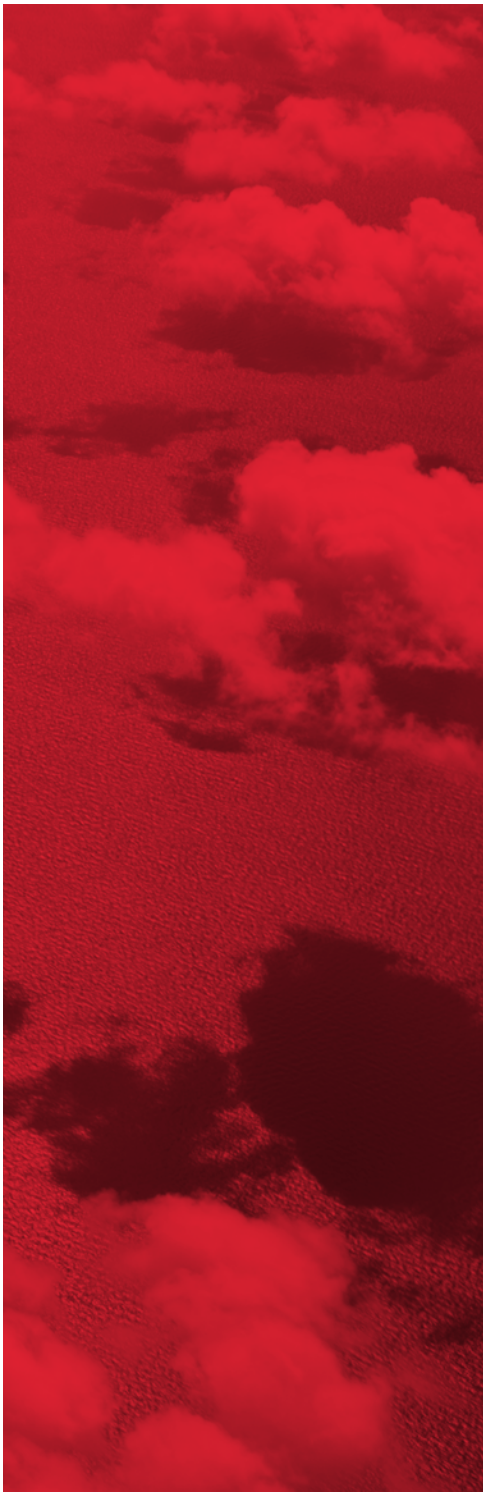
which it is assigned. This optimization, while beneficial for power consumption, is also beneficial economically, as data centers would require less overall power.

In data centers, having mixed criticality systems executing on the same physical medium allows for the creation of new and interesting services. For example, users of the data center might be able to specify which I/O devices they would like to make use of, how many processors they require, and how much memory they should be allocated. Then, the data center would find a physical platform that can guarantee dedicated access to those resources and gives the user exclusive control of those resources. The user's processes might be executing on a machine with several other guest operating systems operating on it, but through the isolation mechanisms of a partitioning hypervisor, the guest would not encounter any interference from them. This dynamic would be very valuable to edge devices, or applications that require strict quality of service guarantees.

Currently, our research is exploring how to make more efficient usage of, and build applications and services for, the environments that a partitioning hypervisor provides. We are also investigating how to better equip Linux with these partitioning features. This will give Linux the ability to be used more optimally in a wider variety of domains and will enable partitioning hypervisors to become more accessible. This accessibility provides an opportunity for a wider audience to explore the potential of partitioning hypervisors.

Linux is a well-known and widely used system, so building these partitioning features into the Linux kernel will not only make them easy to adopt, but will also allow individuals to use Linux to explore their hardware in interesting ways. For example, there are several SBCs (single board computers), such as the UP Squared, that have the hardware one would expect on commodity platforms, but also include GPIOs (general purpose I/O pins).

In some cases, such as Intel's Aero Board, the platforms also include gyroscopes and IMUs. Using partitioning techniques, a core, some memory, and the GPIOs could be isolated and used to emulate something like an Arduino, which can be hosted on the same physical platform as the platform where the Arduino development is occurring. Shared memory channels could then be used to communicate data from the guest where development is occurring to the Arduino-like guest, instead of through a serial link or some other



CRAIG EINSTEIN *is a computer science Ph.D. student at Boston University under the advisement of Professor Richard West. Prior to starting his Ph.D. he received his B.A. in Geophysics and Planetary Sciences in 2016 from Boston University. He researches computer systems with an emphasis on real-time and mixed criticality systems and autonomous control. Upon the completion of his doctorate, he would like to work in the space industry developing systems to make space travel more efficient and accessible.*

communication mechanism. This would increase the bandwidth and flow rate of the data and reduces both the hardware and software complexity of the communication. An example of a potential system configuration for such a system can be seen in **Figure 3**.

With these partitioning features, we can investigate the limits and possibilities of hardware, and discover and develop configurations and protocols to aid in endeavours such as resource and power management, mixed criticality system interactions, and autonomous vehicle dynamics. These features would also provide the flexibility to explore how to allow artificial devices to interact more efficiently with the physical world.

These endeavours and explorations are immediately useful and will become increasingly more prevalent, especially as bigger and harder challenges are tackled, such as creating fully autonomous vehicles, increasing the efficiency and capabilities of our global computing infrastructures, and exploring the universe.

I would like to thank my advisor, Professor Richard West, and Bandan Das for their guidance, mentorship, and insights.

AUTHOR

– Craig Einstein. Ph.D. Student

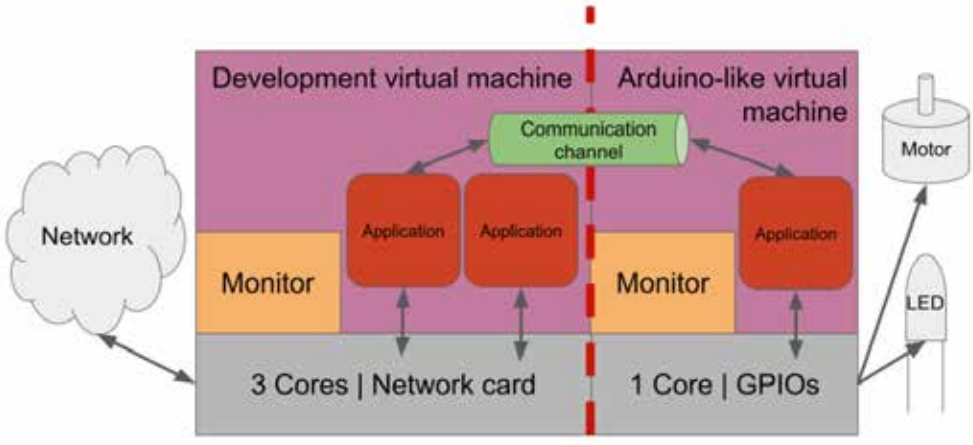


Figure 3. *This figure shows an overview of the hardware and software configuration of an example use-case of a partitioning hypervisor. The regions dedicated to each guest are delineated by the dashed red line.*

RESEARCH PROJECTS UPDATE

Faculty, PhD students, and U.S. Red Hat associates in the Northeast U.S. are collaborating actively on the following research projects. This quarter we highlight collaborative projects at Boston University (BU), Northeastern University, Harvard University, and the University of Massachusetts. We will highlight research colloborations from other parts of the world in future editions of the Research Quarterly. Contact academic@redhat.com from more information on any project.

Academic investigators	Red Hat investigators	Project title
Ali Raza (araza)	Ulrich Drepper, Larry Woodman, Richard Jones	Unikernel Linux

Unikernels are small, lightweight, single address space operating systems with the kernel included as a library within the application. Because unikernels run a single application, there is no sharing or competition for resources among different applications, improving performance and security. Unikernels have thus far seen limited production deployment. This project aims to turn the Linux kernel into a unikernel with the following characteristics: 1) are easily compiled for any application, 2) use battle-tested, production Linux and glibc code, 3) allow the entire upstream Linux developer community to maintain and develop the code, and 4) provide applications normally running vanilla Linux to benefit from unikernel performance and security advantages. The paper Unikernels: The Next Stage of Linux's Dominance was presented at HotOS XVII, The 17th Workshop on Hot Topics in Operating Systems, 2019.

Video presentation

https://www.youtube.com/watch?v=ItvXeolVnVE&feature=youtu.be&t=3h57m40s

Academic investigators	Red Hat investigators	Project title
Tommy Unger, Han Dong, Yara Awad, Prof. Jonathan Appavoo, Prof. Amos Waterland, Prof. Orran Kreiger	Ulrich Drepper	An optimizing operating system: Accellerating execution with speculation

To optimize performance, Automatically Scalable Computation (ASC), a Harvard/BU collaboration attempts to auto-parallelize single threaded workloads, reducing any new effort required from programmers to achieve wall clock speedup. SEUSS takes a different approach by splicing a custom operating system into the backend of a high throughput distributed serverless platform, Apache OpenWhisk. SEUSS uses an alternative isolation mechanism to containers, called Library Operating Systems (LibOSs). LibOSs enable a lightweight snapshotting technique. Snapshotting LibOSs enables two counterintuitive results: 1) although LibOSs inherently replicate system state, SEUSS can cache multiplicatively more functions on a node; 2) although LibOSs can suffer bad "first run" performance, SEUSS is able to reduce cold start times by orders of magnitude. By increasing sharing and decreasing deterministic bringup, SEUSS radically reduces the amount of hardware and cycles required to run a FaaS platform.

Video presentation

https://www.youtube.com/watch?v=h1rpSeaTecQ

Parul Sohal, Prof. Renato Mancuso, Prof. Orran Kreiger	Ulrich Drepper	Removing memory as a noise factor

Academic investigators	Red Hat investigators	Project title
Memory bandwidth is increasingly the bottleneck in modern systems and a resource that, until today, we could not schedule. This means that, depending on what else is running on a server, performance may be highly unpredictable, impacting the 99% tail latency, which is increasingly important in modern distributed systems. Moreover, the increasing importance of high-performance computing applications, such as machine learning and real-time systems, demands more deterministic performance, even in shared environments. Alternatively, many environments resist running more than one workload on a server, reducing system utilization. Recent processors have started introducing the first mechanism to monitor and control memory bandwidth. Can we use these mechanisms to enable machines to be fully used while ensuring that primary workloads have deterministic performance? This project presents early results from using Intel's Resource Director Technology and some insight into this new hardware support. The project also examines an algorithm using these tools to provide deterministic performance on different workloads.		
Video presentation https://www.youtube.com/watch?v=i8JnQ7VKkEM		
Ali Raza (alraza), Prof. Orran Kreiger		"Performance management for serverless computing

Academic investigators	Red Hat investigators	Project title
Serverless computing provides developers the freedom to build and deploy applications without worrying about infrastructure. Resources (memory, cpu, location) specified for a function can affect performance, as well as cost, of a serverless platform, so configuring these resources properly is critical to both performance and cost. COSE uses a statistical learning approach to dynamically adapt the configurations of serverless functions while meeting QoS/SLA metrics and lowering the cost of cloud usage. This project evaluates COSE on a commercial serverless platform (AWS Lambda) as well as in multiple simulated scenarios, proving its efficacy.		
Video presentation https://www.youtube.com/watch?v=7CgBJnNebrQ		

Academic investigators	Red Hat investigators	Project title
Sahil Tikale, Ali Raza (alraza), Leo McGann, Danni Shi, Filip Vukelic, Jacob Daitzman, Prof. Orran Kreiger	Langdon White, Lars Kellog-Stedman, Tzu-Mainn Chen, Gagan Kumar	FLOCX: First Layer of the Open Cloud eXchange
FLOCX provides a marketplace for trading physical servers among co-located pools of hardware where each pool is owned and managed by independent organizations. Using FLOCX, organizations can rent nodes from their co-located neighbors in times of high demand and offer their own resources at a suitable price when others experience high demand. An implementation of FLOCX (https://cci-moc.github.io/flocx/) is running in the Massachusetts Open Cloud environment (https://massopen.cloud).		
Video presentation https://youtu.be/goDpCRLhCao		

Academic investigators	Red Hat investigators	Project title
Han Dong, James Cadden, Yara Awad, Prof. Orran Kreiger, Prof. Jonathan Appavoo	Sanjay Arora	Automatic Configuration of Complex Hardware

A modern network interface card (NIC), such as the Intel X520 10 GbE, is complex, with hardware registers that control every aspect of the NIC's operation from device initialization to dynamic runtime configuration. The Intel X520 datasheet documents over 5600 registers; yet only about 1890 are initialized by a modern Linux kernal. It is thus unclear what the performance impact of tuning these registers on a per application basis will be. In this project, we pursue three goals towards this understanding: 1) identify, via a set of microbenchmarks, application characteristics that will illuminate mappings between hardware register values and their corresponding microbenchmark performance impact, 2) use these mappings to frame NIC configuration as a set of learning problems such that an automated system can recommend hardware settings corresponding to each network application, and 3) introduce either new dynamic or application instrumented policy into the device driver in order to better attune dynamic hardware configuration to application runtime behavior.

Video presentation

https://www.youtube.com/watch?v=8UQTINQTKtQ

Academic investigators	Red Hat investigators	Project title
Emine Ugur Kaynar, Mania Abdi, Prof. Peter Desnoyers, Prof. Orran Kreiger	Matt Benjamin, Brett Niver, Ali Maredia, Mark Kogan	D3N: A multi-layer cache for data centers

Current caching methods for improving the performance of big-data jobs assume abundant (e.g., full bi-section) bandwidth to cache nodes. However, many enterprise data centers and co-location facilities exhibit significant network imbalances due to over-subscription and incremental network upgrades. This project designs and develops D3N, a novel multi-layer cooperative caching architecture that mitigates network imbalances by caching data on the access side of each layer of hierarchical network topology. A prototype implementation, which incorporates a two-layer cache, is highly-performant (can read cached data at 5GB/s, the maximum speed of our SSDs) and significantly improves the performance of big-data jobs. To fully utilize bandwidth within each layer under dynamic conditions, we present an algorithm that adaptively adjusts cache sizes of each layer based on observed workload patterns and network congestion.

Video presentation

https://www.youtube.com/watch?v=troLFFM6btc

Academic investigators	Red Hat investigators	Project title
Ahmed Sanaullah, Prof. Martin Herbordt, Prof. Orran Kreiger	Ulrich Drepper	FPGAs in large-scale computer systems

Secure Multiparty Computation (MPC) is a cryptographic primitive that allows several parties to jointly and privately compute desired functions over secret data. This project developed and deployed JIFF: an extensible general-purpose MPC framework capable of running on web and mobile stacks, showing how developments in distributed systems, web development, and the SMDI paradigm can inform MPC constructs and implementation. JIFF includes a JavaScript library for building applications that rely on secure MPC, with the ability to be run in the browser, on mobile phones, or via Node.js. JIFF is designed so that developers need not be familiar with MPC techniques or know the details of cryptographic protocols in order to build secure applications. This project used JIFF to implement several MPC applications, including a successfully deployed real-world study on economic opportunity for minority-owned businesses in the Boston area and a service for efficient privacy-preserving route recommendation.

Video presentation

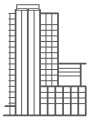
https://www.youtube.com/watch?v=wwwtyIWrTZ0

Academic investigators	Red Hat investigators	Project title
Craig Einstein, Prof. Richard West	Bandan Das	A partitioning hypervisor for latency-sensitive workloads

Quest-V is a separation kernel that partitions services of different criticality levels across separate virtual machines or sandboxes. Each sandbox encapsulates a subset of machine physical resources that it manages without requiring intervention from a hypervisor. In Quest-V, a hypervisor is only needed to bootstrap the system, recover from certain faults, and establish communication channels between sandboxes. The machine physical resources that are given to each sandbox include one or more processing cores, a region of machine physical memory, and a subset of I/O devices. Current Quest-V research is exploring how to manage hardware resources to allow for a power and latency aware system. The partitioning of virtual machines (VMs) onto separate machine resources offers an opportunity for per-sandbox power management. Thus, in idle periods, a sandbox may place its hardware into a suspend state, reducing the power utilization of the sandbox. Depending on the latency and power demands of the sandbox, the sandboxes can be suspended to RAM or to disk. The sandbox can then resume normal power consumption when appropriate. Sandboxes also have the ability to be migrated across hosts to balance system resources and reduce power consumption. This allows for entire machines to be placed into low power states upon the migration of all sandboxes away from those machines. Quest-V is unlike a normal hypervisor in that it allows VMs to suspend and resume individual hardware resources without interfering with the operation of other VMs on the same physical platform. This allows for the creation of systems that are both power and latency aware.

Academic investigators	Red Hat investigators	Project title
Xiaojing Zhu	Sanjay Arora	Code2Vec: Learning code representations

Code2Vec is a neural model for representing snippets of code as fixed-length continuous vectors (code embeddings) that encode some semantic similarities , which enables the application of neural techniques to a wide-range of programming-languages tasks. Embeddings can be applied to performance measurement of program execution in CPU and smarter code completion and finding similar functions in analyzed code. This project analyzed semantic similarities of learned code embeddings parsed from open source python libraries such as numpy, pandas and sklearn. Still in progress is another analysis that learns code embeddings in a supervised manner with the C++ codebase for performance measurement of program execution in CPU with performance counters (e.g. LLC misses to L1 requests, Cycles Per Instruction).



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

RESEARCH QUARTERLY

VOLUME 1:3

COMING IN FEBRUARY:

- **Real-time complex event processing from streaming data**
- **Project Vega: Adding rotation to stellar computational models**
- **Interview with Red Hat's Mark Little on Quarkus and all things Java**

Bringing great research ideas into open source communities

November 2019 – Volume 1: Issue 3

Red Hat