Introducing Research for Practice

EXPERT-CURATED GUIDES TO THE BEST OF CS RESEARCH

Reading a great research paper is a joy. A team of experts deftly guides you, the reader, through the often complicated research landscape, noting the prior art, the current trends, the pressing issues at hand—and then, sometimes artfully, sometimes through seeming sheer force of will, expands the body of knowledge in a fell swoop of 12 or so pages of prose. A great paper contains a puzzle and a solution; these can be useful, enlightening, or both. A great paper is a small, structured quantum of human ingenuity, creativity, and labor, in service of a growing understanding of our world and the future worlds we may inhabit.

Unfortunately, information overload is a defining problem of our time, and computer science research is no exception. The volume of research produced each year in computer science is heartening, but it can be difficult to determine which papers are most deserving of our scarce time. This volume of papers is also at odds with many of the best elements of paper reading: distillation of work to its critical essence, thoughtful consideration of its nuances and the context in which the research was performed, and application of concepts to one's own technical problems and experiences.

As a result, the past few years have seen a rise in interest and organizations—such as Papers We Love and its many chapters—devoted to the joy and utility of reading computer science research: curated-paper discussions have escaped the traditionally academic "reading seminar" format and have been supplanted by groups of hundreds of participants meeting regularly, at startups and community centers, to discuss the latest and greatest computer science research. This is exciting. Why should the greatest of papers be enjoyed only in academia? As a public good, research should be read, discussed, digested, and enjoyed by all interested parties.

ACM has a particularly important role to play in this democratization of access to research. First, the ACM Digital Library is the largest collection of computer science research in the world, with hundreds of thousands of papers, articles, and manuscripts. Second, the ACM membership consists of world experts across all subfields of computer science, from Turing laureates to ACM Fellows, from upstart academics to engineers on the cutting edge of practice. Separately, these are unparalleled resources; put together, they are even more extraordinary.

Research for Practice is born from the potential of this combination. In every RfP column, two experts will introduce a short curated selection of papers on a concentrated, practically oriented topic. Want to learn about the latest and greatest developments in operating systems for data-center workloads? RfP will provide an essential crash course from a world authority by describing the trends in this space, selecting a handful of papers to read, and providing motivation and the critical insights behind each.

This approach is designed to allow you to become fluent in exciting topics in computer science research in a weekend afternoon. In addition, ACM has graciously agreed to provide open access to any Research for Practice paper citations available in the ACM Digital Library. Each installment will cover different topics from different volunteer experts, and we intend to cover the entire range of computer science subfields.

This issue of *acmqueue* magazine contains the first installment of Research for Practice. Were you curious about the data-center operating system trends I just mentioned? You're in luck: *Simon Peter* has a fantastic selection on this topic, including papers on the interplay between emerging I/O subsystems and the kernel, principles for multicore scalability, and systems possibilities for new secure computing hardware. In addition, *Justine Sherry* has contributed an exciting selection on network functions virtualization: our networks are getting smarter, aided by increasingly complex in-network software. This allows functionality beyond traditional network "middlebox" operation, including complex routing and policy deployment and cryptographically secure and private packet processing. Both of these selections highlight practical yet principled research papers. We're especially pleased by how accessible each of our experts has made these otherwise highly technical topics.

Research for Practice is itself an ongoing experiment. We're inspired by the widespread and growing enthusiasm about computer science research as well as the role ACM, its members, and the *acmqueue* readership can play in amplifying this excitement. We welcome your feedback, and please enjoy! —*Peter Bailis*

DATA CENTERS ARE CHANGING THE WAY WE DESIGN SERVER SYSTEMS

BY SIMON PETER

The growing number of cloud service users and volume of data are putting tremendous pressure on I/O, processing, and integrity. Hardware has kept pace: data-center networks allow servers to transmit and receive millions of requests per second with microsecond delivery latencies. An increasing number of processors multiplies server-processing capacities, and new technologies such as Intel's Software Guard Extensions (SGX) help keep sensitive data confidential. As a result, operating systems need to provide these new technologies to applications scalably and efficiently.

The following papers introduce thought-provoking OS design paradigms that address each of these trends. First we attack the I/O performance problem. We then introduce a handy software-interface design rule that ensures that constructed software can scale with the number of processors present in data-center servers. Finally, we learn how to protect the integrity of sensitive data, even from access by the cloud operator. We conclude with an outlook on how these paradigms enable an ecosystem of execution environments for data-center applications.

Dealing with the data deluge

Peter, S., et al. 2014. Arrakis: the operating system is the control plane. *Usenix Symposium on Operating Systems Design and Implementation.* https://www.usenix.org/conference/osdi14/technical-sessions/presentation/peter

Belay, A., et al. 2014. IX: a protected dataplane operating system for high throughput and low latency. *Usenix Symposium on Operating Systems Design and Implementation.* https://www.usenix.org/conference/osdi14/technical-sessions/presentation/belay

These papers discuss the design of operating systems that provide high I/O performance to request-intensive server applications. The authors find that the complexity of monolithic OS kernels is the biggest barrier to server I/O performance and remedy the situation by introducing an I/O model that bypasses the kernel in the common case without losing any of its protection guarantees. Both papers split the OS into a control and a data plane: A kernel-level control plane carries out access control and resource management, while a user-level data plane is responsible for fast I/O mechanisms.

The papers differ in how network I/O policy is enforced. Arrakis reaches for utmost performance by relying on hardware to enforce per-application maximum I/O rates and allowed communication peers. IX trades performance for software control over network I/O, thus allowing the precise enforcement of the I/O behavior of a particular network protocol, such as TCP congestion control.

Both OS models do extremely well supporting an emerging bit of cloud infrastructure: containers. Containers bundle all required components of an application into a manageable unit. Arrakis and IX empower containers to use all I/O capabilities of the underlying server hardware without the overhead of a monolithic OS kernel.

Keeping all processors busy

Clements, A. T., et al. 2013. The scalable commutativity rule: designing scalable software for multicore processors. *ACM Symposium on Operating System Principles.* http://dl.acm.org/citation.cfm?id=2699681 [link to http://queue.acm.org/rfp/vol14iss2.html]

Many OS researchers have worked on the problem of using an increasing number of processor cores to handle growing workload demands. Manually identifying and working around scalability bottlenecks caused by shared resource contention in implementations has often been the answer. This paper asks a different question: Can APIs have an impact on software scalability? The surprising answer is that the impact is not only profound, but also fundamental.

The paper distills its insight into a simple yet effective software-development rule: whenever interface operations commute, they can be implemented in a way that scales. The authors provide a tool that helps developers apply the rule by generating test cases that find scalability bottlenecks in commutative API implementations. They use the tool to evaluate the POSIX API and point out where the API has the ability to scale but its OS implementation hits a bottleneck. They employ the results to develop a new OS that is practically free of scalability bottlenecks.

The scalable commutativity rule applies not just to the design of operating systems, but also to any multicore software system. It should thus be part of the toolkit of any multicore application developer.

Keeping sensitive data confidential

Baumann, A., et al. 2014. Shielding applications from an untrusted cloud with Haven. *Usenix Symposium on Operating Systems Design and Implementation.* https://www.usenix.org/conference/osdi14/technical-sessions/presentation/baumann

Customers trust their cloud providers not to expose any of their data—a tall order, given the staggering complexity of the cloud hardware/software platform. Bugs may easily compromise sensitive data. This paper introduces Haven, a software system that protects the integrity of a program and its data from the entire cloud-execution platform, except for a small trusted block of firmware

To achieve this, Haven uses the recently introduced Intel SGX technology to develop a non-hierarchical OS security model that allows applications to run in a secure region of memory that is protected from outside access, including privileged software such as OS kernels and hypervisors. To support execution on top of an untrusted OS kernel, Haven introduces a mutually distrusting kernel interface that applications access via a user-level library that provides the Windows API.

Haven introduces a new way of protecting data confidentiality. While previous attempts use encryption techniques such as homomorphic encryption to compute on encrypted data in limited cases, Haven relies on hardware-protection technology to address the problem in a more general way.

An ecosystem of application execution environments

These papers establish a new baseline for data-center OS design. Not the traditional Unix model where processes run on top of a shared kernel invoked via POSIX system calls, but protected software containers using scalable library invocations that map directly to hardware mechanisms allow applications to break out of existing OS performance and protection limitations.

This new OS design has the potential to enable an ecosystem of library execution environments that support applications in various ways. For example, a fast library network stack may be linked to a web server to improve its webpage delivery latency and throughput. A Haven-like system call library may be linked to protect the integrity of confidential data held by the application. Finally, a scalable storage stack may be linked to a database to allow it to keep pace with the throughput offered by parallel flash memory. In many cases, these libraries can improve application execution transparently. Together, these new execution environments have the potential to allow applications to match the performance and integrity demands of current and future data-center workloads.

NFV AND MIDDLEBOXES

BY JUSTINE SHERRY

We usually think of networks as performing only one task: delivering packets from sender to receiver. Today's networks, however, do a lot more by deploying special-purpose *middleboxes* to inspect and transform packets, usually to improve performance or security. A middlebox may scan a connection for malicious behavior, compress data to provide better performance on low-resource mobile devices, or serve content from a cache inside the network to reduce bandwidth costs. Both industry and research sources have recently begun to refer to the features implemented by middleboxes as "network functions." Popular open-source network functions include the Snort Intrusion Detection System[3] and the Squid Web Proxy.[4]

To deploy a new network function, a network administrator traditionally purchases a specialized, fixed-function hardware device (the middlebox) implementing, for example, intrusion detection or caching, and physically installs the device at a chokepoint in the network such that all traffic entering or exiting the network must pass through it. Alternatively, an administrator might use an off-the-shelf server as a middlebox, installing software such as Snort, Squid, or a proprietary software package, and then routing traffic through the server at a chokepoint in the network.

NFV (network functions virtualization) is a new movement in networking that takes the software-based approach to an extreme. The NFV ISG (industry specification group) envisions a future in which all middlebox functionality is implemented in software.[2] Network administrators will deploy a server or cluster of servers dedicated to network functions, and network virtualization software will automatically route traffic through various network functions.

NFV promises many benefits for network administrators. It reduces costs by moving from special-purpose to general-purpose hardware, makes upgrades as easy as a software patch, offers the opportunity to scale on demand, and promises more efficient installations with multiple network functions potentially sharing a single server, leaving few resources wasted. NFV has tremendous momentum in the networking community—the NFV working group has more than 200 industrial members[1]—but is in its infancy and was founded only in late 2012.

Here we present three highlights from the research community on middleboxes and NFV, and conclude by discussing some of the challenges and opportunities that

NFV presents for application developers.

What capabilities do network functions implement?

Carpenter, B., Brim, S. 2002. *Middleboxes: taxonomy and issues. RFC 3234, IETF.*
https://tools.ietf.org/html/rfc3234

Though it predates NFV by about a decade, this article remains a nice summary of the features for which middleboxes are commonly deployed. The document could have gone into more depth about application-layer behaviors such as exfiltration detection or intrusion detection—increasingly common in today's corporate networks—but these behaviors are more common today than they were in 2002 when the article was written. Nonetheless, it remains the most comprehensive survey of middlebox functionality to date, and most of the features it describes remain in common use.

What does an NFV-managed network look like?

Palkar, S., Lan, C., et al. 2015. E2: a framework for NFV Applications. *ACM Symposium on Operating Systems Principles.*
http://dl.acm.org/citation.cfm?id=2815423

This article provides the cleanest vision for an NFV-managed cluster to date. The authors describe a system called E2, which automatically schedules and configures network functions on a cluster of general-purpose servers. E2 allows a network administrator to specify a "configuration" (e.g., all traffic on port 80 should be routed through this HTTP proxy, all traffic to this subnet should be processed by an IDS), and the framework will automatically instantiate software instances and a routing configuration to ensure that the policy is met. E2 is conceptually similar to cloud frameworks such as OpenStack or RightScale but in practice involves many different technical challenges, including scheduling to ensure that bandwidth is not overutilized, ensuring low latency, and enabling efficient communication and "chaining" between network functions.

Can I control how network functions process my traffic?

Naylor, D., et al. 2015. Multi-Context TLS (mcTLS): enabling secure in-network functionality in TLS. *ACM SIGCOMM.* *http://dl.acm.org/citation.cfm?id=2787482* *[link to http://queue.acm.org/rfp/vol14iss2.html]*

Sherry, J., et al. 2015. BlindBox: deep packet inspection over encrypted traffic. *ACM SIGCOMM.* *http://dl.acm.org/citation.cfm?id=2787502* *[link to http://queue.acm.org/rfp/vol14iss2.html]*

Today, application developers have no way of controlling which network functions process their traffic, short of making a phone call to their network administrators. Nonetheless, developers may have concerns about inspection or modification of traffic sent by their applications—especially with regard to privacy. Hence, many developers choose to encrypt their entire connection (e.g., using SSL/TLS). While this preserves privacy, it also prevents *all benefits* of middlebox processing. These two articles propose new cryptographic protocols, mcTLS and BlindBox, that would let application developers allow certain middlebox operations but restrict others. The two articles propose very different approaches to the same problem and are worth reading side by side.

What does NFV mean for application developers?

As NFV makes the deployment and configuration of network functions/middleboxes easier, application developers can expect to see increasingly complex behavior from their networks. While this capability for complex behavior retains some of the old challenges of middleboxes (e.g., privacy), it also introduces a huge new opportunity for application developers. NFV enables application developers to run and execute their code not only on end hosts they maintain, but also in the network itself.

For example, a developer who designs a custom load-balancing filter based on a unique service architecture might write the new code to run on the load balancer itself. A web service may implement a custom cache to serve encrypted content to its users, deploying the in-network cache within its customers' ISPs within virtual machines hosted in the provider's infrastructure. With the ability to execute *arbitrary* code in the network—and smart routing and scheduling to ensure that the right traffic receives such processing—NFV opens an entirely new programming platform for developers. The next big app store may be for features deployed within data-center networks, ISPs, or even on home routers.

References

1. NFV ISG. List of members; https://portal.etsi.org/TBSiteMap/NFV/NFVMembership.aspx.
2. NFV ISG. 2012. Network functions virtualization: an introduction, benefits, enablers, challenges, and call for action; https://portal.etsi.org/NFV/NFV_White_Paper.pdf.
3. Snort; https://www.snort.org/.
4. Squid; http://www.squid-cache.org/.

LOVE IT, HATE IT? LET US KNOW feedback@queue.acm.org

Peter Bailis *will join Stanford University as an assistant professor of computer science, after spending the academic year visiting MIT CSAIL. He received a Ph.D. in computer science from UC Berkeley in 2015 and an A.B. in computer science from Harvard in 2011. His research in the Future Data Systems group (http://futuredata.stanford.edu/) focuses on the design and implementation of next-generation data-intensive systems.*

Justine Sherry *is a doctoral candidate at UC Berkeley. Her interests are in computer networking; her work includes middleboxes, networked systems, measurement, cloud computing, and congestion control. Sherry's dissertation focuses on new opportunities and challenges arising from the deployment of middleboxes—such as firewalls and proxies—as services offered by clouds and ISPs. She received an M.S. from*

UC Berkeley in 2012 and a B.S. and B.A. from the University of Washington in 2010. She is a National Science Foundation Graduate Research Fellow, has won paper awards from both Usenix NSDI and ACM SIGCOMM, and is always on the lookout for a great cappuccino.

Simon Peter *is an assistant professor at the University of Texas at Austin, where he leads research in operating systems and networks. He received a Ph.D. in computer science from ETH Zurich in 2012 and an M.S. in computer science from the Carl-von-Ossietzky University in Oldenburg, Germany, in 2006. Before joining UT Austin in 2016, he was a research associate at the University of Washington from 2012-2016. For his work on high-I/O-performance operating systems, he received the Jay Lepreau best paper award (2014) and the Madrona prize (2014). He has conducted further award-winning systems research at various locations, including MSR SVC and Cambridge, Intel Labs, and UC Riverside.*

CONTENTS