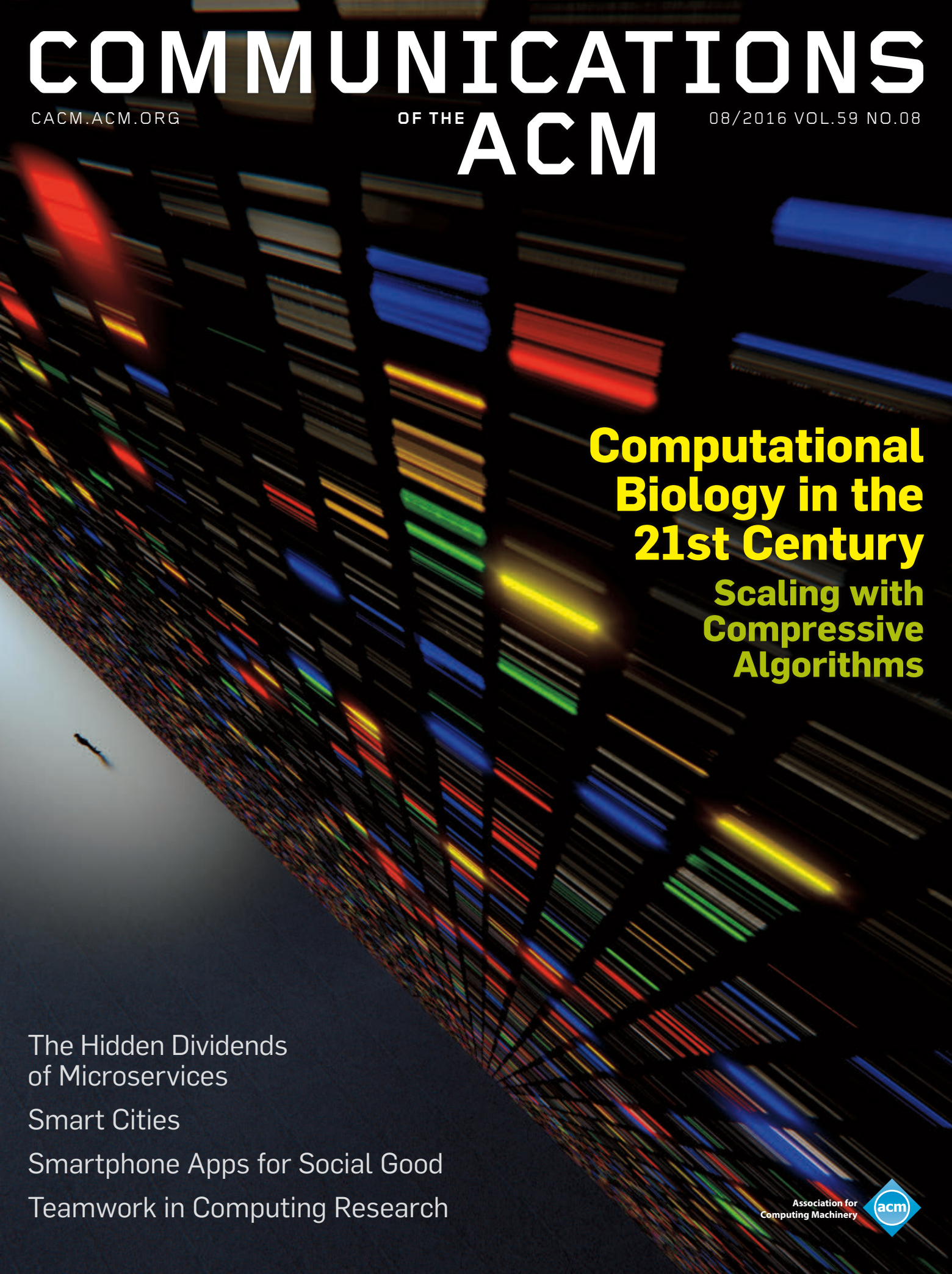
COMMUNICATIONS OF THE ACM

CACM.ACM.ORG

08/2016 VOL.59 NO.08



Computational Biology in the 21st Century

Scaling with Compressive Algorithms

The Hidden Dividends of Microservices

Smart Cities

Smartphone Apps for Social Good

Teamwork in Computing Research

Association for Computing Machinery

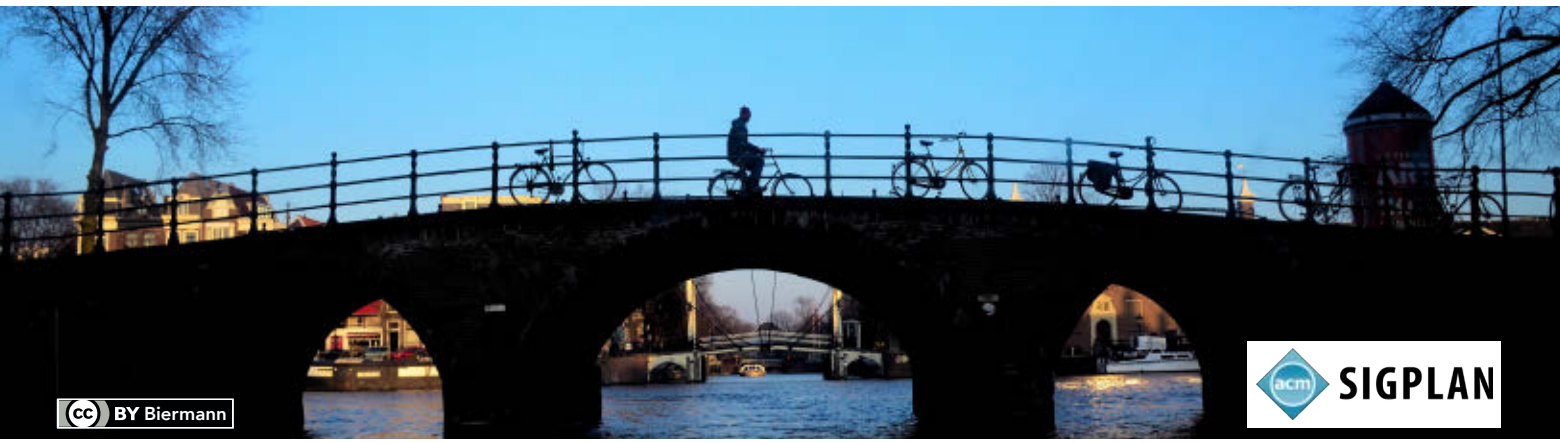
this.splash 2016

Amsterdam

Sun 30 October – Fri 4 November 2016

ACM SIGPLAN Conference on Systems, Programming, Languages and Applications: Software for Humanity (SPLASH)

OOPSLA	Novel research on software development and programming
Onward!	Radical new ideas and visions related to programming and software
SPLASH-I	World class speakers on current topics in software, systems, and languages research
SPLASH-E	Researchers and educators share educational results, ideas, and challenges
DLS	Dynamic languages, implementations, and applications
GPCE	Generative programming: concepts and experiences
SLE	Principles of software language engineering, language design, and evolution



SPLASH General Chair: Eelco Visser

OOPSLA Papers: Yannis Smaragdakis

OOPSLA Artifacts: Michael Bond, Michael Hind

Onward! Papers: Emerson Murphy-Hill

Onward! Essays: Crista Lopes

SPLASH-I: Eelco Visser, Tijs van der Storm

SPLASH-E: Matthias Hauswirth, Steve Blackburn

DLS: Roberto Ierusalimschy

Workshops: Jan Rellermeyer, Craig Anslow

SLE General Chair: Tijs van der Storm

SLE Papers: Emilie Balland, Daniel Varro

GPCE General Chair: Bernd Fischer

GPCE Papers: Ina Schaefer

Student Research Competition: Sam Guyer, Patrick Lam

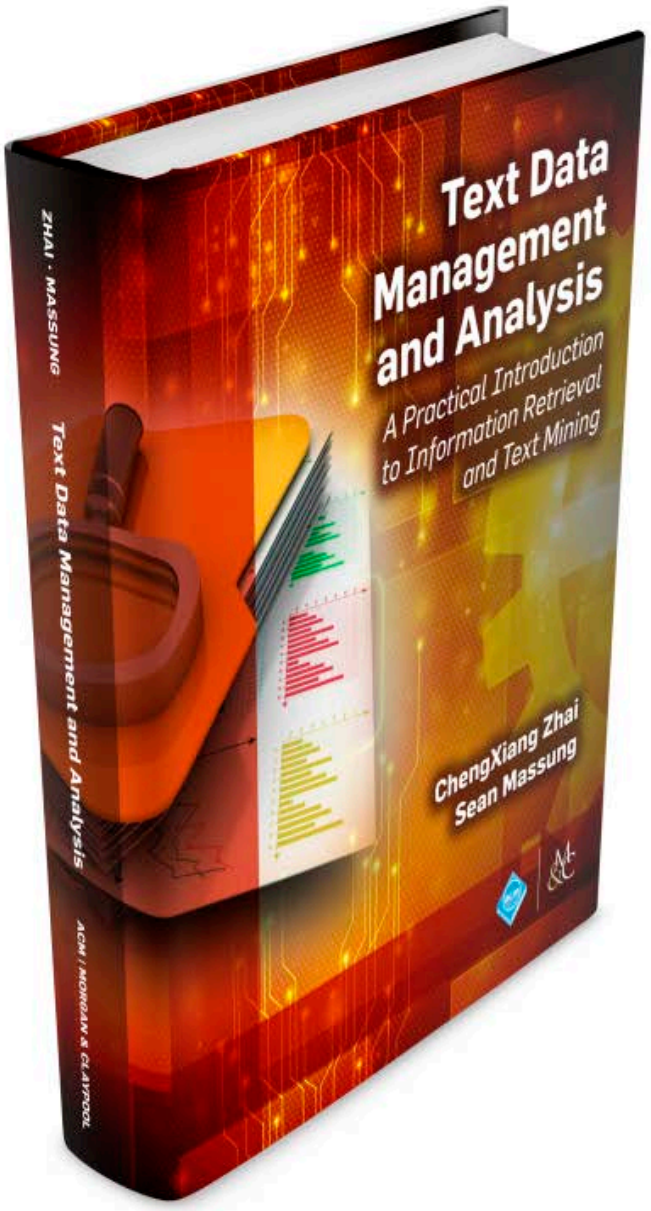
Posters: Jeff Huang, Sebastian Erdweg

Publications: Alex Potanin

Publicity and Web: Tijs van der Storm, Ron Garcia

Student Volunteers: Daco Harkes

@splashcon 2016.splashcon.org bit.ly/splashcon16



The most useful and practical knowledge for building a variety of text data applications.

CS STUDENTS (Undergrad & Graduate)
LIBRARY & INFORMATION SCIENTISTS
TEXT DATA PRACTITIONERS

ChengXiang Zhai & Sean Massung (Authors)
University of Illinois at Urbana-Champaign

Text Data Management and Analysis covers the major concepts, techniques, and ideas in **information retrieval** and **text data mining**. It focuses on the practical viewpoint and **includes many hands-on exercises designed with a companion software toolkit** (i.e., MeTA) to help readers learn how to apply techniques of information retrieval and text mining to real-world text data. It also shows readers how to **experiment** with and **improve** some of the algorithms for interesting application tasks. The book can be used as a **text** for computer science undergraduates and graduates, library and information scientists, or as a **reference** for practitioners working on relevant problems in **managing and analyzing text data**.

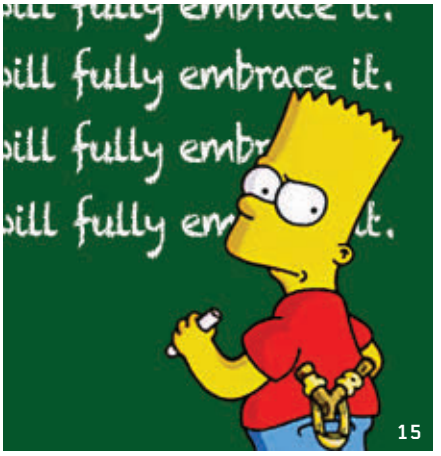
http://books.acm.org
http://www.morganclaypoolpublishers.com/text

COMMUNICATIONS OF THE ACM

Departments

Last Byte

News



15

Viewpoints

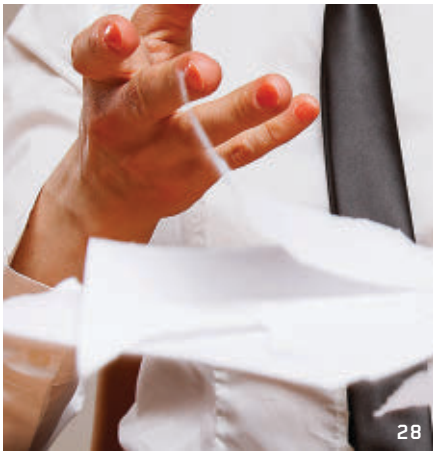
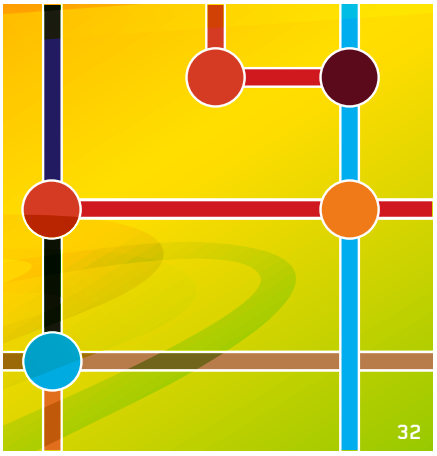

28

PHOTO BY MARYNA PLESHKUN

Association for Computing Machinery
Advancing Computing as a Science & Profession

Practice



Articles' development led by acmqueue
queue.acm.org

About the Cover: Researchers today are able to amass unprecedented amounts of biological data, a state that offers as many challenges as opportunities. This month's cover story explores ways to harness this explosion of data in order to analyze it in meaningful, valuable ways. Cover illustration by Peter Crowther Associates.

IMAGES FROM SHUTTERSTOCK.COM

Contributed Articles

Review Articles

Watch the authors discuss their work in this exclusive *Communications* video. http://cacm.acm.org/videos/computational-biology-in-the-21st-century

Research Highlights

Watch the author discuss his work in this exclusive *Communications* video. http://cacm.acm.org/videos/ur-web

COMMUNICATIONS OF THE ACM

Trusted insights for computing's leading professionals.

Communications of the ACM is the leading monthly print and online magazine for the computing and information technology fields. *Communications* is recognized as the most trusted and knowledgeable source of industry information for today's computing professional. *Communications* brings its readership in-depth coverage of emerging areas of computer science, new trends in information technology, and practical applications. Industry leaders use *Communications* as a platform to present and debate various technology implications, public policies, engineering challenges, and market trends. The prestige and unmatched reputation that *Communications of the ACM* enjoys today is built upon a 50-year commitment to high-quality editorial content and a steadfast dedication to advancing the arts, sciences, and applications of information technology.

ACM, the world's largest educational and scientific computing society, delivers resources that advance computing as a science and profession. ACM provides the computing field's premier Digital Library and serves its members and the computing profession with leading-edge publications, conferences, and career resources.

Subscriptions
An annual subscription cost is included in ACM member dues of $99 ($40 of which is allocated to a subscription to *Communications*); for students, cost is included in $42 dues ($20 of which is allocated to a *Communications* subscription). A nonmember annual subscription is $269.

ACM Media Advertising Policy
Communications of the ACM and other ACM Media publications accept advertising in both print and electronic formats. All advertising in ACM Media publications is at the discretion of ACM and is intended to provide financial support for the various activities and services for ACM members. Current advertising rates can be found by visiting http://www.acm-media.org or by contacting ACM Media Sales at (212) 626-0686.

Single Copies
Single copies of *Communications of the ACM* are available for purchase. Please contact acmhelp@acm.org.

COMMUNICATIONS OF THE ACM
(ISSN 0001-0782) is published monthly by ACM Media, 2 Penn Plaza, Suite 701, New York, NY 10121-0701. Periodicals postage paid at New York, NY 10001, and other mailing offices.

POSTMASTER
Please send address changes to
Communications of the ACM
2 Penn Plaza, Suite 701
New York, NY 10121-0701 USA

Printed in the U.S.A.

Association for Computing Machinery

DOI:10.1145/2967359

Vicki L. Hanson

From the New ACM President

I thank all of you who recently sent along good wishes upon my election to ACM president. Many also asked "What do you want to accomplish during your presidency?"

To begin to answer this, I refer back to the recent message from outgoing President Alex Wolf as he looked back on his two years in office (*Communications*, June 2016, p. 5). He noted that the foundations of ACM are in excellent shape; we have committed volunteers, dedicated headquarters staff, and a reliable revenue stream. I thank him for his efforts in guiding ACM to this position of strength. My goals are to build on this foundation, being responsive to changing demographics, finding new ways to add value to both our student and professional members, and balancing the desire for open access to our Digital Library with the need to maintain our economic health.

While there are over two million global students and professionals who benefit from our publishing, professional development, education curricula, worldwide conferences, and networking opportunities, most of these individuals do not join ACM. All of our work, however, is only possible through the volunteer efforts and the organizational stability of membership. We need to reconsider the benefits of membership in a professional society. For this, we will be seeking input in the coming months to clarify what our community most values.

One demographic that has been slow to join ACM is early career professionals. I will be spearheading efforts to increase the involvement of this cohort. In rapidly changing academic, entrepreneurial, practitioner, and global computing environments, networking through ACM can provide real value. In the coming months, new initiatives for these young professionals will be formulated and announced.

In terms of publications, I look forward to our Digital Library continuing to evolve. Under the direction of ACM's Publications Board, this community resource will be moving from a static repository of documents and videos to one that includes computing artifacts such as data and software as well as new services. The Digital Library underpins ACM's financial stability but the drive toward open access will continue to challenge assumptions about the exchange of information and data in the scientific community. Questions of how ACM as an academic publisher will derive needed funding in this changing content and services environment will continue to dominate discussions with multiple stakeholders.

In my candidate statement, I mentioned that ACM's efforts related to increased diversity and inclusion need to be fully woven into our core fabric. The recent election of the first all-female team of ACM Executive Officers is an opportunity to highlight the contributions that women have made to computing and to inspire young women to view computing as a career. While we can be proud of this milestone, we must be mindful that diversity involves many issues and professionals at all career stages. We will expand the scope for our diversity efforts, including accessibility and diversity concerns globally.

In rapidly changing academic, entrepreneurial, practitioner, and global computing environments, networking through ACM can provide real value.

ACM must also continue to serve the worldwide computing community in all our activities. Our publishing, professional development, conferences, and other efforts must not just be available to global professionals, but also must take into account varying regional priorities. I am deeply committed to the fostering of this global perspective.

This year marked the 50th anniversary of the ACM A.M. Turing award. We can be deservedly proud of the honorees who have made such profound impacts on technology and society. Throughout the coming year we will be celebrating the achievements of these pioneers. More information on these celebrations will be forthcoming.

In sum, during my presidency I will work to develop new initiatives to ensure that ACM provides ever-greater value to the global computing community. Over the next two years, I look forward to meeting many of you and exploring how, together, we can achieve this.

Vicki L. Hanson (vlh@acm.org) is ACM President, Distinguished Professor at Rochester Institute of Technology, and a professor at the University of Dundee. Twitter: @ACM_President

SHAPE THE FUTURE OF COMPUTING.
JOIN ACM TODAY.

ACM is the world's largest computing society, offering benefits and resources that can advance your career and enrich your knowledge. We dare to be the best we can be, believing what we do is a force for good, and in joining together to shape the future of computing.

SELECT ONE MEMBERSHIP OPTION

ACM PROFESSIONAL MEMBERSHIP:

- ❑ Professional Membership: $99 USD
- ❑ Professional Membership plus ACM Digital Library: $198 USD ($99 dues + $99 DL)
- ❑ ACM Digital Library: $99 USD (must be an ACM member)

ACM STUDENT MEMBERSHIP:

- ❑ Student Membership: $19 USD
- ❑ Student Membership plus ACM Digital Library: $42 USD
- ❑ Student Membership plus Print *CACM* Magazine: $42 USD
- ❑ Student Membership with ACM Digital Library plus Print *CACM* Magazine: $62 USD

❑ **Join ACM-W:** ACM-W supports, celebrates, and advocates internationally for the full engagement of women in all aspects of the computing field. Available at no additional cost.

Priority Code: CAPP

Payment Information

Name

ACM Member #

Mailing Address

City/State/Province

ZIP/Postal Code/Country

Email

Payment must accompany application. If paying by check or money order, make payable to ACM, Inc., in U.S. dollars or equivalent in foreign currency.

❑ AMEX ❑ VISA/MasterCard ❑ Check/money order

Total Amount Due

Credit Card #

Exp. Date

Signature

Purposes of ACM

ACM is dedicated to:

1) Advancing the art, science, engineering, and application of information technology
2) Fostering the open interchange of information to serve both professionals and the public
3) Promoting the highest professional and ethics standards

Return completed application to:
ACM General Post Office
P.O. Box 30777
New York, NY 10087-0777

Prices include surface delivery charge. Expedited Air Service, which is a partial air freight delivery service, is available outside North America. Contact ACM for more information.

Satisfaction Guaranteed!

BE CREATIVE. STAY CONNECTED. KEEP INVENTING.

Association for Computing Machinery

1-800-342-6626 (US & Canada)
1-212-626-0500 (Global)

Hours: 8:30AM - 4:30PM (US EST)
Fax: 212-944-1318

acmhelp@acm.org
acm.org/join/CAPP

DOI:10.1145/2964341

Vinton G. Cerf

Star Struck in Lindau

Among the innovations pioneered by John White during his years as CEO of ACM was a new relationship with the Klaus Tschira Foundation that sponsors the Heidelberg

Laureate Forum[a] [HLF] in the third quarter of each year. The attendees include about 200 math or computer science students and recipients of the mathematics Fields Medal, the Nevanlinna Prize, the Abel Prize, and ACM's A.M. Turing award for computer science. I have had the pleasure of attending the first three meetings of the HLF. Since 1951, however, there has been an annual meeting of Nobel laureates[b] with support from several organizations including the aforementioned Klaus Tschira Foundation. The HLF is patterned after the Nobel meeting: students meet with a collection of participating laureates. It was decided last year to link these two events by having a Nobel laureate address the participants of the HLF and to have an HLF laureate address the participants of the Nobel annual meeting. As I write this column, I am in the midst of the Nobel meeting in Lindau and, frankly, star struck at meeting people whose names have been in the news for their extraordinary work. I have the honor to be the first of the HLF participants to address the Nobel meeting and if you think this is not a daunting prospect, think again!

My theme, however, is not just the amazing collection of brilliance at both meetings but the clear and increasing role that computing and computer science are playing in basic research. To be sure, computers have long been utilized to make predictions based on theory and to assess measurements to determine whether the theoretical predictions match the experimental evidence. The 2013 Nobel Prize in Chemistry, however, underscored the potential for using computers to predict biochemical interactions on the basis of computer-based modeling. Three chemists won the Nobel Prize in Chemistry in 2013[c] "for multiscale models for complex chemical systems." As I meet other basic and experimental scientists at this week-long event, I am struck by how often computers play a key role in discovery science. That our discipline touches almost everything in the scientific world is not an overstatement.

I met Martin Karplus at this event. He was among the 2013 Nobel laureates and opened my eyes to the incredible complexity of processes found inside cells. This is not a simple sea of liquid in which organelles and chemicals are swimming. The cell actually has a very complex structure, including so-called microtubules that help to guide the transport of molecules where they are needed in the cell. They are like highways inside the cell. There are molecular robots (I am not kidding!) that literally travel along these highways, carrying specific molecules to targets in the cell. This looks sort of like Multiprotocol Label Switching guiding packets along optical channels! Moreover, like the Internet, there are "addresses" to indicate where the robots are supposed to deliver the molecules. The cells encompass a molecular package switching system (ok, bad pun)! I have not yet learned enough about the way in which the addressing is done, but it is apparently multi-layered and even has the property that a molecule can be targeted to get stuck in an intracellular membrane in addition to being delivered to a target organelle.

The cell uses DNA and RNA to guide the production of proteins by assembling amino acids and by delivering them on the intracellular highway to their destinations. The DNA is used to produce so-called "messenger RNA" that is used in the process of generating proteins using the cell's ribosomes to interpret the RNA and to assemble the prescribed molecular structure. Of course Theorem 206[d] applies here and I have left out the roles of transfer-RNA and ribosomal-RNA in this process. As we learn more about the internal functionality of cells, and succeed in modeling their operation, we will shed additional light on the plausible paths by which this functionality has evolved. Research is under way to discover by experiment the simplest form of replicating cells and to use computer models to help postulate plausible origins of life.

It would be difficult to overstate the excitement I feel for the role our discipline is playing in surveying the scientific discovery landscape and modeling the processes we find to understand the "how" and maybe the "why" of their operation. This underscores for me and I hope for you, the importance of advancing the state of the art of computing and its capacity to analyze and model the complex phenomena that we are finding all around us at the micro and the macro levels. C

a http://www.heidelberg-laureate- forum.org/
b http://www.lindau-nobel.org/
c http://www.nobelprize.org/nobel_prizes/chemistry/laureates/2013/; Martin Karplus, Arieh Warshel, and Michael Levitt.
d Theorem 206: "Everything is more complicated."

Vinton G. Cerf is vice president and Chief Internet Evangelist at Google. He served as ACM president from 2012–2014.

DOI:10.1145/2963167

Future Cyberdefenses Will Defeat Cyberattacks on PCs

HIGHLY CLASSIFIED INFORMATION on personal computers in the U.S. today is largely protected from the attacks Daniel Genkin et al. described in their article "Physical Key Extraction Attacks on PCs" (June 2016). Here, I outline cost-effective defenses that will in the future completely defeat such attacks, while making even stronger cyberattacks extremely difficult.

For example, tiny Faraday cages can be constructed in a processor package so encryption/decryption can be performed without the possibility of inadvertent emanations that could be measured or exploited, because all external communication to a cage would be through optical fiber and the cage's power supply is filtered.[1] This way, encryption keys and encryption/decryption processes would be completely protected against the attacks described in the article. In such a Faraday cage, advanced cryptography (such as Learning With Errors) could not be feasibly attacked through any known method, including quantum computing.

Hardware can likewise help protect software, including operating systems and applications, through RAM-processor package encryption. All traffic between a processor package and RAM can be encrypted using a Faraday cage to protect a potentially targeted app (which is technically a process) from operating systems and hypervisors, other apps, and other equipment, including baseband processors, disk controllers, and USB controllers. Even a cyberattack that compromises an entire operating system or hypervisor would permit only denial of service to its applications and not give access to any application or related data storage. Similarly, every-word-tagged extensions of ARM and X86 processors can be used to protect each Linux kernel object and each Java object in an app from other such objects by using a tag on each word of memory that controls how memory can be used. Tagged memory can make it much more difficult for a cyberattacker to find and exploit software vulnerabilities because compromising a Linux kernel object or a Java application object does not automatically give power over any other object. Security-access operations by such objects can be strengthened through "inconsistency robustness" providing technology for valid formal inference on inconsistent information.[1] Such inconsistency robust inference is important because security-access decisions are often made on the basis of conflicting and inconsistent information.

Such individual processor-package cyberdefense methods and technologies will make it possible, within, say, the next five years, to construct a highly secure board with more than 1,000 general-purpose coherent cores, greatly diminishing dependence on datacenters and thereby decreasing centralized points of vulnerability.[1]

These technologies promise to provide a comprehensive advantage over cyberattacks. The U.S., among all countries, has the most to lose through its current cyberdefense vulnerabilities. In light of its charter to defend the nation and its own multitudinous cyberdefense vulnerabilities, the Department of Defense is the logical agency to initiate an urgent program to develop and deploy strong cyberdefenses using these technologies and methods. Industry and other government agencies should then quickly incorporate them into their own operations.

Reference

1. Hewitt, C. *Security Without IoT Mandatory Backdoors: Using Distributed Encrypted Public Recording to Catch & Prosecute Suspects.* Social Science Research Network, June 16, 2016; http://papers.ssrn.com/sol3/papers.cfm?abstract_id=2795682

Carl Hewitt, Palo Alto, CA

Authors Respond:

We agree that high-security designs can mitigate physical side channels. However, they are typically difficult to design, expensive to manufacture, reduce performance, and are unavailable in most commercial and consumer contexts. Moreover, their security depends on understanding the feasible attacks. Faraday cages, optical couplers, and power filters would not stop acoustic leakage through vents or self-amplification attacks that induce leakage at frequencies below the filter's design specification. The problem of creating inexpensive hardware or software that adequately mitigates all feasible side-channel attacks thus remains open.

Daniel Genkin, Lev Pachmanov, Itamar Pipman, Adi Shamir, and Eran Tromer, Tel Aviv, Israel

Fewer Is Better Than More Samples When Tuning Software

The emphasis on visualizing large numbers of stack samples, as in, say, flame graphs in Brendan Gregg's article "The Flame Graph" (June 2016) actually works against finding some performance bottlenecks, resulting in suboptimal performance of the software being tuned. Any such visualization must necessarily discard information, resulting in "false negatives," or failure to identify some bottlenecks. For example, time can be wasted by lines of code that happen to be invoked in numerous places in the call tree. The call hierarchy, which is what flame graphs display, cannot draw attention to these lines of code.[1] Moreover, one cannot assume the bottlenecks can be ignored; even a particular bottleneck that starts small does not stay small, on a percentage basis, after other bottlenecks have been removed. Gregg made much of 60,000 samples and how difficult they are to visualize. However, he also discussed finding and fixing a bottleneck that resulted in saving 40% of execution time. That means the fraction of samples displaying the bottleneck was at least 40%. The bottleneck would thus have been displayed, with statistical certainty, in a purely human examination of 10 or 20

random stack samples—with no need for 60,000. This is generally true of any bottleneck big enough, on a percentage basis, to be worth fixing; moreover, every bottleneck grows as others are removed. So, if truly serious performance tuning is being done, it is not necessary or even helpful to visualize thousands of samples.

Reference
1. Dunlavey, M.R. Unknown events in nodejs/v8 flamegraph using perf_events; http://stackoverflow.com/a/27867426/23771

Michael R. Dunlavey, Needham, MA

Author Responds:

More samples provide more benefits. One is that performance wins of all magnitudes can be accurately quantified and compared, including even the 1%, 2%, and 3% wins, finding more wins and wasting less engineering time investigating false negatives. Samples are cheap. Engineering time is not. Another benefit is quantifying the full code flow, illustrating more tuning possibilities. There are other benefits, too. As for code invoked in numerous places, my article discussed two techniques for identifying them—searching and merging top-down.

Brendan Gregg, Campbell, CA

When Designing APIs Take the User's View

I doubt many software developers were surprised by Brad A. Myers's and Jeffrey Stylos's conclusions in their article "Improving API Usability" (June 2016). I anecdotally suspect most poor API designs are based on what the implementation does rather than what the user needs.

API designers know the implementation too well, suffering from the curse of knowledge. The API makes sense to them because they understand the context. They do not consider how an API can be confusing to users without it. Worse yet, implementation details could leak into the API without the designer noticing, since the delineation of the API and feature implementation might be blurred.

I agree with Myers and Stylos that the API should be designed first. Design and code operational scenarios to confirm API usability before diving too deeply into feature design and implementation. Spend time in the mind-set of the user. The scenario code used to confirm the API can also be reused as the foundation of test code and user examples.

Myers and Stylos recommended avoiding patterns. Creation patterns involve API complications beyond usability. Gamma et al.[1] defined creation-method names that suggest how the returned object was constructed; for example, factory-, singleton-, and prototype-created objects are acquired through `create`, `instance`, and `clone`, respectively. Not only do these different object-acquisition-method names create more API confusion, they violate encapsulation, since they suggest the creation mechanism to a user. This naming constraint limits the designer's ability to switch to a different design-pattern mechanism. Gamma et al. also failed to define creation-pattern clean-up mechanisms that are critical in C++, one of their example languages.

I prefer to unify my creation-pattern objects by defining `acquire` to acquire an object and `release` to release an object when it is no longer being used. These unification methods encapsulate the creation pattern and provide for clean up, as well. They yield a consistent API based on user need rather than on an implementation-creation mechanism.

They also affect usability, however, since they are not standard nomenclature, so I take it one step further for APIs used by others. I use the Pointer to IMPLementation, or PIMPL, idiom to create a thin API class that wraps the functionality. The API class calls `acquire` in its constructor and `release` in its destructor. Wrapping the unification methods within the constructor and destructor results in an API that uses the source language's natural creation syntax yet still offers the advantages of the creation design patterns within the underlying implementation.

Reference
1. Gamma, E., Helm, R., Johnson, R., and Vlissides, J. *Design Patterns.* Addison-Wesley, Reading, MA, 1995.

Jim Humelsine, Neptune, NJ

Communications welcomes your opinion. To submit a Letter to the Editor, please limit yourself to 500 words or less, and send to letters@cacm.acm.org.

BLOG@CACM

The *Communications* Web site, http://cacm.acm.org, features more than a dozen bloggers in the BLOG@CACM community. In each issue of *Communications*, we'll publish selected posts or excerpts.

twitter
Follow us on Twitter at http://twitter.com/blogCACM

DOI:10.1145.2950111 http://cacm.acm.org/blogs/blog-cacm

Inside the Great Wall

During a trip to China, Jason Hong watches for signs of new technologies.

Jason Hong
The Emerging Technology Landscape in China
http://bit.ly/1Rtm2SR
May 25, 2016

I have been in China about two weeks so far, and have been amazed at how much the technology landscape has changed since I last spent time here in 2008. I wanted to share some of these observations because they represent some really unique and compelling uses of technology.

WeChat Mobile Social Networking

Perhaps the most obvious change is the ubiquity of WeChat, a combination of mobile chat and social networking app. Pretty much every Chinese person I met used WeChat, and I saw people using it in restaurants, on the subway, waiting for the bus, pretty much everywhere. WeChat was developed by powerhouse Tencent, and its features are basically the same as any social networking site: you can add friends, post status updates that friends can see, and send messages to friends. The app feels much cleaner than Facebook's mobile app, though, probably because WeChat started mobile first and as a result had to be very simple.

A cool feature of WeChat is the ability to send virtual red envelopes to others. Real red envelopes are given by close friends and family members on birthdays and Chinese New Year, and contain money. I remember always looking forward to getting these envelopes as a kid. WeChat's virtual envelopes can also be used to transfer money to others, and this feature is already very popular for these special occasions. There is even a Wikipedia page about these virtual red envelopes (http://bit.ly/1r4gwAc) reporting over a billion were sent last year for Chinese New Year.

Mobile Payment Systems

Another obvious change is the widespread adoption of mobile payment systems. Many people who hosted us for dinner used their smartphones to pay, just by taking a picture of a QR code on the receipt. One person told us how you can use AliPay (from Alibaba, a major e-commerce company), WeChat, or several other apps to buy things from vending machines, again through QR codes. The fact they were compatible was really cool, since I would have expected companies to try to do proprietary systems.

Another person told us how he forgot his wallet at home one day. He almost turned around, but decided to see how well he could do just with his smartphone. It turns out he did perfectly fine, even being able to buy food from street vendors without any problems.

I found the popularity of mobile payment in China amazing, especially since I have yet to directly see anyone in the U.S. use any smartphone mobile payment apps (other than Square, which addresses a different payment space). I have only heard secondhand stories from a few of students at my university about using Venmo or other payment apps.

My best explanation is *relative advantage*. In the U.S., credit cards are already very popular, and so mobile payment apps only offer a small advantage in this context. In contrast, in China, credit cards were not very common, and so mobile payment systems conveniently filled in the gap, allowing China to leapfrog ahead.

Taxi Hailing Systems

Didi is a very popular app for hailing taxis. It is only about four years old, and in about half the taxis I was in, the drivers had a holder for their smartphone and were actively using it. One driver even had his app in some kind of active mode, where you could see

and hear every single nearby request, and it was going off about every 20 seconds. (I should also note driving safety is a dodgy concept in China. Half the taxis I was in didn't have safety belts, pedestrians walk anytime across the street, and motorcycles often go against traffic.)

Didi used to be two competing taxi hailing companies, but they merged a short while back. I do not know much about Chinese law, but there do not seem to be antitrust laws to prevent large mergers like this. Some people I talked to also said taxi hailing was much better before the merger, since the two companies offered heavy discounts, making it very cheap to get around, but no longer.

Didi is also really interesting when compared with Uber's business model. Unlike Uber, which takes a cut out of the fare, Didi does not seem to; instead, people told me Didi analyzes and sells the data from riders. For example, they can determine origin and destination pairs easily, making it possible to infer demographics and interests of riders. They can then sell that data and even use it for highly targeted advertisements or offers. I was told Didi recently closed a major deal with some banks to do some offers. Apple also announced it will be investing US$1 billion into Didi, perhaps to help with Apple Pay in China, to improve their maps, and to sell other software and services.

It is also not clear to me how well Uber can do in China. Uber was reported to have lost about $1 billion in China last year because of its heavy discounts to win market share. It is not clear to me if Uber China also works with taxis, but Uber's standard model of having everyday people act as drivers might not work as well in China, since the density of taxis in major Chinese cities is already pretty high.

Another interesting story I heard is many of these companies are suffering from fraud. For example, one friend told me how Uber was essentially paying riders to use their service, and how one person had 100 smartphones and colluded with drivers to do fake rides to cash in. In fact, a lot of people commented how all of these discounts were essentially VCs fueling the economy, since a lot of these popular apps were offering deep cuts in prices to gain market share.

Compounding things is the fact that a lot of similar companies exist. Many friends I talked with observed that if you come up with a good startup idea, it does not take very long for copycat companies to appear and offer their VC-funded discounts. One person commented how there used to be a dozen companies that would wash your car overnight (for about 20 RMB, or about $3), but now it has dropped to about two. So competition is very stiff, and it is going to be exciting to see how things play out in the next few years.

Same- or Next-Day Delivery

If you walk on a major street of any Chinese city, you will likely see a dozen motorized bikes with logos like TMall, JD.com, or even Amazon. Same- or next-day delivery is very common and convenient in China. My wife and I used these services to buy diapers and clothes, and they deliver right to your door. You can also pay on delivery, probably because of the lack of online payment infrastructure, though the mobile payment systems described earlier can also be used.

The density of cities in China is one clear reason these delivery services work. One VC explained another non-obvious reason: in the U.S., retail infrastructure is mostly saturated, meaning e-commerce offers limited relative advantage over what already exists, and often works by cannibalizing brick-and-mortar stores. In contrast, retail infrastructure in China is not as well established, meaning e-commerce has a lot more room for potential growth. These factors, along with relatively low labor and delivery costs, make same- or next-day delivery in China very effective in practice.

Other Odds and Ends

A few more observations about the tech landscape in China that did not fit in the above categories. One friend told me how the iPhone was sometimes called the "kidney" because one teen literally sold his kidney so he could buy an iPhone; he figured he had two kidneys and really wanted an iPhone. More surprisingly, I was told that for many people, the iPhone costs as much as two months of salary, but I still saw Apple's smartphone almost everywhere.

Chinese people also have a strong affinity for numbers. One interesting quirk is that most telephone numbers in China do not have any dashes or spaces, just a string of 10 or 11 digits. One colleague explained Chinese people have a strong capacity for remembering numbers. However, I can assure readers this is not genetic, given my lack of this facility.

More interestingly, many "holidays" have been created based on special numbers. You may know how Chinese people avoid the number 4 (since it sounds like the word "death" in Chinese) and really like the number 8 (since it sounds like "good fortune"). Riffing on this idea, May 20 (5/20) has been advanced by companies as a new kind of Valentine's Day, since if you say the individual digits it sounds a little bit like "I love you." Yeah, it is sort of a stretch, but WeChat has taken full advantage of this by making it easy to send 520 RMB (about $75) to one's significant other; there is no option for 5.20 or 52.00 RMB. From what I was told, 5/20 was invented just a few years ago, but it has already caught on.

November 11 is another new "holiday," representing single people (11/11, or all 1's). Students I talked to said it was very similar to Black Friday in the U.S., the day of frenzied shopping after Thanksgiving, because what better way is there to celebrate being single than going shopping? Apparently, Alibaba has been breaking sales records every year on this date.

Parting Thoughts

I hope I have conveyed some of the energy and excitement I have been seeing the past few weeks in China. Even though I am supposed to be on vacation, it is hard to miss all of the innovation and advances everyday people are making use of. I am really looking forward to seeing what the next decade of change will bring. C

Jason Hong is an associate professor in the School of Computer Science at Carnegie Mellon University.

Science | DOI:10.1145/2949662 Marina Krakovsky

Reinforcement Renaissance

The power of deep neural networks has sparked renewed interest in reinforcement learning, with applications to games, robotics, and beyond.

IMAGE BY MAXUSER

Each time DeepMind has announced an amazing accomplishment in game-playing computers in recent months, people have taken notice.

First, the Google-owned, London-based artificial intelligence (AI) research center wowed the world with a computer program that had taught itself to play nearly 50 different 1980s-era Atari games—from Pong and Breakout to Pac-Man, Space Invaders, Boxing, and more—using as input nothing but pixel positions and game scores, performing at or above the human level in more than half these varied games. Then, this January, DeepMind researchers impressed experts with a feat in the realm of strategy games: AlphaGo, their Go-playing program, beat the European champion in the ancient board game, which poses a much tougher AI challenge than chess. Less than two months later, AlphaGo scored an even greater victory: it won 4 games in a best-of-5 series against the best Go player in the world, surprising the champion himself.

The idea that a computer can learn to play such complex games from scratch and achieve a proficient level elicits gee-whiz reactions from the general public, and DeepMind's triumphs have heightened academic and commercial interest in the AI field behind DeepMind's methods: a blend of deep neural networks and reinforcement learning called "deep reinforcement learning."

"Reinforcement learning is a model of learning where you're not given a solution—you have to discover it by trial and error," explains Sridhar Mahadevan, a professor at the University of Massachusetts Amherst, a longtime center of research into reinforcement learning.

The clearest contrast is with supervised learning, the kind used to train image recognition software, in which the supervision comes in the form of labeled examples (and requires people to label them). Reinforcement learning, on the other hand, "is a way of not needing labels, or labeling automatically by who's winning or losing—by the rewards," explains University of Alberta computer scientist Rich Sutton, a co-founder of the field of reinforcement learning and co-author of the standard textbook on the subject. In reinforcement learning, the better your moves are, the more rewards you get, "so you can learn to play the Go game by playing the moves and winning or losing, and no one has to tell you if that was a good move or a bad move because you can figure it out for yourself; it led to a win, so it was a good move."

Sutton knows the process is not as simple as that. Even in the neat, tightly controlled world of a game, deducing which moves lead to a win is a notoriously difficult problem because of the delay between an action and its reward, a key feature of reinforcement learning. In many games, you receive no feedback at all until the end of the game, such as a 1 for a win or a –1 for a loss.

"Typically, you have to go through hundreds of actions before your score increases," explains Pieter Abbeel, an associate professor at the University of California, Berkeley, who applies deep reinforcement learning to robotics. (For a robot, a reward comes for completing a task, such as correctly assembling two Lego pieces.) "Before you understand how the game works, and are learning through your own trial and error," Abbeel says, "you just kind of do things, and every now and then your score goes up, and every now and then it goes down, or doesn't go up. How do you tease apart which subset of things that you did contributed to your score going up, and which subset was just kind of a waste of time?"

This thorny question—known as the credit assignment problem—remains a major challenge in reinforcement learning. "Reinforcement learning is unique in that it's the only machine learning field that's focused on solving the credit assignment problem, which I think is at the core of intelligence," says computer scientist Itamar Arel, a professor of electrical engineering and computer sciences at the University of Tennessee and CEO of Osaro, a San Francisco-based AI startup. "If something good happens now, can I think back to the last *n* steps and figure out what I did that led to the positive or negative outcome?"

Despite the buzz around DeepMind, combining reinforcement learning with neural networks is not new.

Just as for human players, figuring out smart moves enables the program to repeat that move the next time it faces the same situation—or to try a new, possibly better move in hope of stumbling into an even higher reward. In extremely small worlds (think of a simple game like Tic-Tac-Toe, also known as Noughts and Crosses), the same exact situations come up again and again, so the learning agent can store the best action for every possible situation in a lookup table. In complex games like chess and Go, however, it is impossible to enumerate all possible situations. Even checkers has so much branching and depth that the game yields a mind-boggling number of different positions. So imagine what happens when you move from games to real-world interactions.

"You're never going to see the same situation a second time in the real world," says Abbeel, "so instead of a table, you need something that understands when situations are similar to situations you've seen before." This is where deep learning comes in, because understanding similarity—being able to extract general features from many specific examples—is the great strength of deep neural networks. These networks, whose multiple layers learn relevant features at increasingly higher levels of abstraction, are currently the best available way to measure similarities between situations, Abbeel explains.

The two types of learning—reinforcement learning and deep learning through deep neural networks—complement each other beautifully, says Sutton. "Deep learning is the greatest thing since sliced bread, but it quickly becomes limited by the data," he explains. "If we can use reinforcement learning to automatically generate data, even if the data is more weakly labeled than having humans go in and label everything, there can be much more of it because we can generate it automatically, so these two together really fit well."

Despite the buzz around DeepMind, combining reinforcement learning with neural networks is not new. TD-Gammon, a backgammon-playing program developed by IBM's Gerald Tesauro in 1992, was a neural network that learned to play backgammon through reinforcement learning (the TD in the name stands for Temporal-Difference learning, still a dominant algorithm in reinforcement learning). "Back then, computers were 10,000 times slower per dollar, which meant you couldn't have very deep networks because those are harder to train," says Jürgen Schmidhuber, a professor of artificial intelligence at the University of Lugano in Switzerland who is known for seminal contributions to both neural networks and reinforcement learning. "Deep reinforcement learning is just a buzzword for traditional reinforcement learning combined with deeper neural networks," he says.

Schmidhuber also notes the technique's successes, though impressive, have so far been in narrow domains, in which the current input (such as the board position in Go or the current screen in Atari) tells you everything you need to know to guide your next move. However, this "Markov property" does not normally hold outside of the world of games. "In the real world, you see just a tiny fraction of the world through your sensors," Schmidhuber points out, speaking of both robots and humans. As humans, we complement our limited perceptions through selective

memories of past observations; we also draw on decades of experience to combine existing knowledge and skills to solve new problems. "Our current reinforcement learners can do this in principle, but humans still do it much better," he says.

Researchers continue to push reinforcement learners' capabilities, and are already finding practical applications.

"Part of intelligence is knowing what to remember," says University of Michigan reinforcement learning expert Satinder Singh, who has used the world-building game Minecraft to test how machines can choose which details in their environment to look at, and how they can use those stored memories to behave better.

Singh and two colleagues recently co-founded Cogitai, a software company that aims to use deep reinforcement learning to "build machines that can learn from experience the way humans can learn from experience," Singh says. For example, devices like thermostats and refrigerators that are connected through the Internet of Things could continually get smarter and smarter by learning not only from their own past experiences, but also from the aggregate experiences of other connected devices, and as a result become increasingly better at taking the right action at the right time.

Osaro, Arel's software startup, also uses deep reinforcement learning, but promises to eliminate the costly initial part of the learning curve. For example, a computer learning to play the Atari game Pong from scratch starts out completely clueless, and therefore requires tens of thousands of plays to become proficient—whereas humans' experience with the physics of balls bouncing off walls and paddles makes Pong intuitive even to children.

"Deep reinforcement learning is a promising framework, but applying it from scratch is a bit problematic for real-world problems," Arel says. A factory assembling smartphones, for example, requires its robotic manufacturing equipment to get up to speed on a new design within days, not months. Osaro's solution is to show the learning agent what good performance looks like "so it gets a starting point far better than cluelessness," enabling the agent to rapidly improve its performance.

Even modest amounts of demonstration, Arel says, "give the agent a head start to make it practical to apply these ideas to robotics and other domains where acquiring experience is prohibitively expensive." C

Further Reading

Mnih, V., et al.
Human-level control through deep reinforcement learning. *Nature* (2015), vol. 518, pp. 529–533.

Silver, D., et al.
Mastering the game Go with deep neural networks and tree search. *Nature* (2016), vol. 529, pp. 484–489.

Tesauro, G.
Temporal difference learning and TD-Gammon. *Communications of the ACM* (1995), vol. 38, issue 3, pp. 58–68.

Sutton, R.S., and Barto, A.G.
Reinforcement Learning: An Introduction. MIT Press, 1998, https://mitpress.mit.edu/books/reinforcement-learning

Based in San Francisco, **Marina Krakovsky** is the author of *The Middleman Economy: How Brokers, Agents, Dealers, and Everyday Matchmakers Create Value and Profit* (Palgrave Macmillan, 2015).

Milestones

Computer Science Awards, Appointments

NEWEST AMERICAN ACADEMY MEMBERS INCLUDE SIX COMPUTER SCIENTISTS

Among the 176 new Fellows recently elected to the American Academy of Arts and Scientists are six in the computer science arena:

▸ Jeffrey A. Dean, Google. A Google Senior Fellow, Dean is a Fellow of ACM, and shared the ACM-Infosys Foundation Award for 2012 with Sanjay Ghemawat.

▸ Sanjay Ghemawat, Google. A research scientist who works with MapReduce and other large distributed systems, Ghemawat is also the author of the popular calendar application iCal, and with Dean, wrote the 2004 paper "MapReduce: Simplified Data Processing on Large Clusters."

▸ Anna R. Karlin, University of Washington. The Microsoft Professor of Computer Science & Engineering at the University of Washington, Karlin, an ACM Fellow since 2012, writes about the use of randomized packet markings to perform IP traceback, competitive analysis of multiprocessor cache coherence algorithms, unified algorithms for simultaneously managing all levels of the memory hierarchy, Web proxy servers, and hash tables with constant worst-case lookup time.

▸ Tom M. Mitchell, Carnegie Mellon University (CMU). Chair of the Machine Learning Department at CMU and E. Fredkin University Professor, Mitchell has contributed to the advancement of machine learning, artificial intelligence, and cognitive neuroscience.

▸ Tal D. Rabin, IBM T.J. Watson Research Center. Head of the cryptography research group at theWatson Research Center, Rabin's research focuses on the design of efficient, secure encryption algorithms, as well as secure distributed algorithms, the theoretical foundations of cryptography, number theory, and the theory of algorithms and distributed systems.

▸ Scott J. Shenker, University of California, Berkeley. Leader of the Initiatives Group and Chief Scientist of the International Computer Institute, Shenker received the 2002 SIGCOMM for lifetime contribution to the field of communication networks "For contributions towards an understanding of resource sharing on the Internet." He has been an ACM Fellow since 2003.

SCHNEIDER RECEIVES CRA SERVICE AWARD

The Computing Research Association (CRA) has named Fred Schneider, Samuel B. Eckert Professor and Chair of Computer Science at Cornell University, recipient of the Service to CRA Award for his ongoing work with the organization.

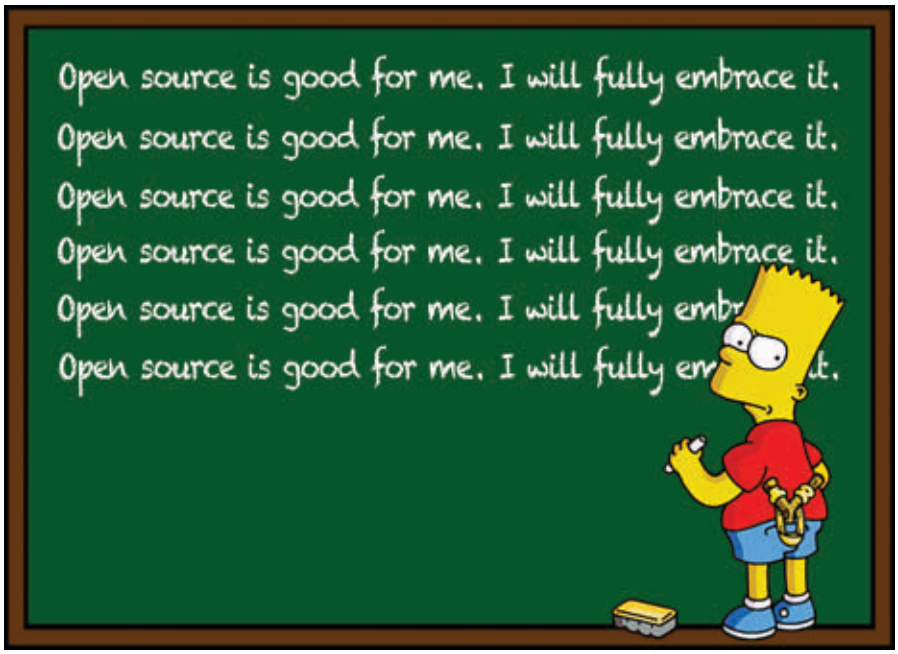
A member of the CRA Board from 2007 to 2016, Schneider served as chair of the organization's Government Affairs Committee for seven years, helping to drive CRA's policy agenda, and developing the Leadership in Science Policy Institute to educate computing researchers on how science policy in the U.S. is formulated.

Schneider led the organization's Committee on Best Practices for Hiring, Promotion, and Scholarship in conducting interviews with more than 75 academic and industry computing and information unit heads to understand issues and gain insights from practice. Preliminary recommendations were vetted with department chairs and CRA Deans at the CRA Conference at Snowbird in 2014, and were published in a CRA Best Practices memo, "Incentivizing Quality and Impact: Evaluating Scholarship in Hiring, Tenure, and Promotion."

Technology | DOI:10.1145/2949684 Gary Anthes

Open Source Software No Longer Optional

Open development and sharing of software gained widespread acceptance 15 years ago, and the practice is accelerating.



In the spring of 1991, a 21-year-old Finnish student named Linus Torvalds sat down to write code that would ultimately revolutionize the world of software development. In a Usenet newsgroup post late that summer, he told the world about his work: "I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones," he wrote. "This has been brewing since April and is starting to get ready. I'd like to know what features most people would want. Any suggestions are welcome, but I won't promise I'll implement them :-)."

Indeed, users of his Linux operating system have wanted a lot of features over the past quarter-century, and Torvalds has not had to add them himself. Linux today has more than 18 million lines of source code and some 12,000 participating developers. There are tens of millions of Linux users worldwide, from owners of Android smartphones to corporate data center managers to scientists at supercomputer centers.

It is a remarkable success story by almost any measure, and much of that success owes to a model of software development called "open source." There are slightly different definitions of the idea, but essentially open source refers to software that is publicly available as source code and may be freely used, modified, and redistributed, without charge for the license (but sometimes with a charge for the service of distributing the software).

History

Software offered to the public in open source format is not a new idea. Beginning in the 1950s, a user group called SHARE, working with IBM, published applications and utilities as source code for use on IBM mainframes. In the 1970s, AT&T Bell Labs licensed Unix source code to government and academic researchers pretty much without restriction.

The modern open source software (OSS) movement got its start with Richard Stallman, a staunch "free software" advocate, who wrote an open source text editor called Emacs in 1976. In 1983, he launched the GNU Project to develop a free Unix-like operating system and related utilities. At the same time he also founded the non-profit Free Software Foundation to promote wide collaboration in the development, distribution, and modification of free software, including GNU Project software such as GNU Emacs and the GCC C compiler.

The creation and use of OSS grew steadily after 1983, but users and developers, especially large corporations, often have looked at it askance. "Who would trust a mission-critical application to software written by a bunch of long-haired anarchists?" users asked. "Why would you give away software created at great expense to competitors?" developers wanted to know. William Scherlis, a computer science professor at Carnegie Mellon University and director of the university's Institute for Software Research, says these objections are no longer valid, if they ever were.

Scherlis says the view of the open source movement as a combination of anarchy and demagoguery is a myth. "It looks like hundreds or thousands of people contributing from around the world, but generally it's a very small core group that has intellectual ownership of the code base," he says. "There's usually a hierarchical structure so that control is maintained, and the major successful projects, like Apache and Eclipse, have elaborate ownership and governance structures: the Apache Foundation, the Eclipse Foundation, and so on."

Allison Randal, president of the Open Source Initiative, which advo-

cates for OSS and maintains a list of industry-standard OSS licenses, says the open source community essentially declared victory in 2010, by which time she says the tide of opinion had flowed overwhelmingly from proprietary software to OSS. She cites a recent survey of 1,300 IT professionals by Black Duck Software that showed the percentage of companies running part or all of their operations on OSS had almost doubled between 2010 and 2015, from 42% to 78%. The number reporting they contribute to open source projects rose from 50% in 2014 to 64% last year, she adds.

Why Do It?

"It comes down to economic necessity," says Randal, who is also a development manager at Hewlett Packard Enterprise. "If nobody else was using open source, you could ignore it, but if others use it, they are getting something free and you are not, and they have an advantage. You can't be a startup in Silicon Valley and not use it."

There are also reasons why large companies like HP increasingly use OSS, Randal says. "They get bogged down by their massive patent portfolios. OSS is a good way for them to innovate because they can pool their patents." For example, she says, the 500 companies that participate in the OpenStack project for cloud computing, including AT&T, IBM, and Intel, agree to license their patents to the neutral, non-profit OpenStack Foundation and thereby to all OpenStack users and contributors. "The companies have agreed not to attack each other with patent disputes around their collaborative work," Randal says. "It's a safe space for all of them to work in."

Last year, Google surprised some observers by releasing for general use the source code for its TensorFlow software, a set of tools for developing deep learning applications, including neural networks. It is the AI engine behind various Google apps—such as Google Photos, which can identify objects in pictures it has never seen before—but the code previously was off limits to external parties wanting to develop such apps.

Work on a predecessor system for deep learning, called DistBelief, began four years ago and resulted in a development and production tool for Google, but it was not flexible enough for others to adapt to their purposes, says Google senior fellow Jeffrey Dean. "If you wanted to do a more exotic neural network, some of those were hard," he says. TensorFlow was developed from the start to be open sourced and was written to minimize its dependencies on other internal Google tools and libraries.

Dean says Google traditionally has published the ideas behind its technologies in journals. By open sourcing TensorFlow, Google has gone further by making it easier for others to try Google's ideas and code in their own software. That will enable users to try different machine-learning techniques, spawning advances that may help Google in return. "We hope that a whole community will spring up around this, and we will get a wide variety of contributors, from students and hobbyists to large companies," Dean says.

For years, Microsoft lagged behind many other developers in its embrace of OSS. In a speech in 2001, Microsoft senior vice president Craig Mundie said, "The OSS development model leads to a strong possibility of unhealthy 'forking' of a code base, resulting in the development of multiple incompatible versions of programs, weakened interoperability, product instability, and hindering businesses ability to strategically plan for the future ... It has inherent security risks and can force intellectual property into the public domain ... [OSS] isn't successful in building a mass market and making powerful, easy-to-use software broadly accessible to consumers."

Allison Randal, president of the Open Source Initiative, says the open source community essentially declared victory in 2010.

Yet in 2004, Microsoft dipped its giant toe into the OSS waters with the release of the open source Windows Installer XML Toolset (WiX). In 2005, Microsoft open sourced the F# programming language and, soon after, a number of other things. Last year, it released the open source development framework and runtime system .NET Core, a free implementation of its .NET Framework, for Windows, Linux, and Mac OS X.

Microsoft now participates in more than 2,000 open source projects, says Anders Hejlsberg, a technical fellow and a lead developer of the open source tools C# and TypeScript. "New projects today are open source by default, unless there are good reasons why they shouldn't be," he says. "That's a complete switch from the proprietary mind-set of earlier days."

Microsoft is collaborating with Google on the development of Google's next version of Angular, the popular Web user-interface framework. The open source project will combine features of Angular with features from Microsoft's TypeScript, a superset of JavaScript. Says Hejlsberg, "Previously it was, 'Those are our competitors; we can't work with them.' But now it's, 'Oh, gosh, they are trying to solve problems we've already solved, and vice versa. We should work together for the benefit of both companies and the community at large.'"

While altruism toward the external development community sometimes plays a part, the open sourcing of .NET Core was mostly a financial decision, says Randal of the Open Source Initiative. ".NET is pretty old, and they hit a point where they realized they'd get more value out of releasing it as open source, getting a lot of eyes on the code, and getting contributions back."

Carnegie Mellon's Scherlis says recent open source projects have shown an increased focus on software assurance. "With OSS we have a chance to do better at providing users with not just code, but evidence of quality." That might take the form of test cases, performance evaluations, code analyses, or inspection reports, he notes.

Downsides

Scherlis cautions not to get swept away with open source euphoria; major efforts like Apache may have tight governance, but some projects do not. He points to the devastating Heartbleed security bug discovered in the OpenSSL library in 2014 as an example; the bug left an estimated 500,000 trusted computers vulnerable to breaches of cryptographic security. "OpenSSL wasn't a well-funded consortium, it was just a small group doing it," Scherlis says. "But it was so good and so essential, everybody used it."

Scherlis says users are dreaming when they think, "Many eyes have cast their gaze upon this code, and so it is good." He explains, "It's possibly true for shallow bugs, but not so much for the deep bugs that vex all projects—the global quality properties of the system, architectural flaws, concurrency problems, deep security problems, timing and performance problems, and so on."

Finally, Scherlis warns OSS is not really "free." In reality, most users will pay someone to adapt the software to work in their data centers and will incur internal support and maintenance costs for it. If the software is mission critical, the company will want to devote staff to the external open source project to ensure its needs are met over time.

OSS has become big business since Stallman started the Free Software Foundation. The company GitHub has become the go-to place for developers and users of open software, from large companies like Apple, Google, and Microsoft to thousands of startups. According to GitHub's Brandon Keepers, the company hosts 31 million open source projects used by 12 million developers.

Keepers, GitHub's head of open source software, commends Apple for the way it released in 2014 its open source programming language Swift. "The way they did their launch was one of the most impressive we've seen," Keepers says. "They invited the community into the process."

That is the wave of the future as developers take open source more and more seriously, Keepers predicts. "We are seeing companies treating open source launches like product launches. They want to make a big splash, but they want to make sure there is support for the project after the launch. So you have not just coders, you have community managers, marketing teams, and product managers looking at what is the experience of users coming to the project." C

"New projects today are open source by default, unless there are good reasons why they shouldn't be. That's a complete switch from the proprietary mindset of earlier days."

Further Reading

Charny, B.
Microsoft Raps Open-Source Approach, *CNET News*, May 3, 2001
http://www.cnet.com/news/microsoft-raps-open-source-approach/

Google
Interviews with Google's Angular team about their use of Microsoft's open source TypeScript
https://www.youtube.com/watch?v=hvYnjJc88OI

Kon, F. and Souza, B.
The Open-Source Ecosystem, *Open Source Initiative, 2012*
http://flosscc.org/spread-the-word [video]

Meeker, H.
Open (Source) for Business: A Practical Guide to Open-Source Software Licensing, *CreateSpace Independent Publishing Platform*, April 6, 2015
http://www.amazon.com/exec/obidos/ASIN/1511617772/flatwave-20

Stallman, R.
Free Software, Free Society: Selected Essays, 3rd ed., *Free Software Foundation*, Oct. 2015
http://shop.fsf.org/product/free-software-free-society-3-paperback/

Gary Anthes is a technology writer and editor based in Arlington, VA.

ACM Member News

MAKING SOFTWARE SAFER, RATHER THAN FASTER

Michael Franz, a professor in the computer science department at the University of California, Irvine, is co-inventor of the trace compilation technology that eventually became the JavaScript engine in Mozilla's Firefox browser. He has spent much of his career making software go faster.

Franz says his "original background" is in programming languages and compilers.

He recalls reading a book on data structures and algorithms by Niklaus Wirth, the designer of Pascal and recipient of the 1984 ACM A.M. Turing Award, then writing him a letter saying, "this would be interesting" and asking "what can I do to study under you?" He received a response from the university registrar's office, Franz recalls, "saying they had taken the liberty to enroll me in their series of entrance examinations, and asked could I be there in two weeks. I passed, and that is how I got the opportunity to study computer science at ETH Zurich, the Swiss Federal Institute of Technology."

Today, Franz sees computer security as a more immediate problem. As a result, he has been helping to pioneer artificial diversity. "The idea behind artificial diversity is to generate multiple versions of a program, so an attacker doesn't know which version is running, making it far more difficult to exploit."

During the last few years, Franz says he has been focused on making things safer, rather than faster, although not at the expense of performance. "We invent a security feature, which creates a drag, and then we put on our other hat and optimize the drag away again. So in the end, we have a new security feature which is almost performance-neutral."

—John Delaney

Society | DOI:10.1145/2949664 Logan Kugler

Smartphone Apps for Social Good

Mobile apps make it easier, faster, and cheaper to create massive impact on social causes ranging from world hunger to domestic violence.

The internet is chock-full of gripes about Millennials, the smartphone-obsessed generation that reached young adulthood at the turn of the century. Millenials are entitled, lazy, self- —and selfie- —absorbed, and uninterested in the world at large. They vex and puzzle employers in equal measure, and they cannot be counted on to do anything other than, well, whatever they feel like doing.

Tell that to a new generation of app makers who are busy building programs that make it easy and fun to do massive good around the world. Their apps feed the hungry, clothe the naked, and shelter the homeless, all with a tap of that little screen typically reserved for Angry Birds or Amazon purchases.

The first wave of smartphone apps—the Instagrams, Foursquares, and Snapchats of the world—prided themselves on being social. This new generation of apps prides itself on being socially good, and they are being adopted most frequently by Millennials.

Apps that do social good range from ethical marketplaces like Orange Harp, which "makes the world more socially conscious and sustainable by providing people access to amazing products and behind-the-scenes details about how they are made," to Feedie, which donates a meal to the non-profit Lunchbox Fund each time a user shares pictures of his or her own food at participating restaurants.

Far from wasting their time ordering stuff, broadcasting their breakfast plans, or gaming with friends, Millennials are using apps like these that do social good to change the world, because they are conditioned to do so, says Steve de Brun, co-creator of MicroHero, a free app that allows users to earn money for charity by taking online surveys.

The Lunchbox Fund fosters education by providing nourishing meals to children in rural areas of South Africa.

"Smartphone users are ordering meals, cars, making appointments, and conducting more and more aspects of their personal and work lives from their devices," de Brun says. "Why wouldn't they also be able to donate, give back, or effect social change from their phones as well?"

To all the Millennial naysayers out there, it might be time to revise your criticisms: the Millennial generation is connected, conscientious, and ready to combat social ills.

Charity Meets Cool

Perhaps Millennials like to pat themselves on the back a little too much (we are only human). The human race has always had an altruistic streak; it is not exclusive to recent generations. Yet the youth of today have a few advantages that help them do more good faster, says Anbu Anbalagapandian, who works for the Orange Harp ethical mobile marketplace.

"With advances in technology, it is much easier to create solutions and have a bigger impact," says Anbalagapandian. "For example, donating food from a local restaurant to a homeless shelter, or microlending to a small business in remote parts of the world, have been made incredibly easy."

PHOTO COURTESY OF THE LUNCHBOX FUND

Orange Harp does what a decade ago might have been fiscally or physically impossible: it connects consumers with hand-picked fashion producers who use only the most ethical processes and materials. The app wears its rejection of unsustainable manufacturing like a badge of honor, broadcasting its work with small businesses that are good for people and the planet. The company goes so far as to condemn the "modern-day slavery" that defines the garment industry.

That approach seems to garner acclaim. The Orange Harp app is highly rated in Apple's App Store, with consumers enthusiastic about knowing their dollars are being spent ethically. It was rated by Apple as one of the "best new apps" in 2015.

Orange Harp understands its millennial audience well: to do good, you have to look good, because if you look good, your message gets attention. If your message gets attention, it can spread like wildfire.

"Information spreads much faster now," says Anbalagapandian. "We can use social media to create awareness around problems that affect our people and planet."

"Information spreads much faster now," says Anbalagapandian. Social media can "create awareness around problems that affect our people and planet."

This is not a trend confined to apps that funnel money and donations to good causes.

SafeNight is an app that connects victims of abuse to hotel rooms when shelters are full. Smartphones are the ideal solution; rooms are needed in near-real time, and time is of the essence in domestic violence situations.

Project Noah is an app nature lovers can use to catalog animals and plants they encounter; the database is then used for conversation efforts.

VolunteerMatch connects aspiring do-gooders with opportunities to contribute their time to a good cause in their local area.

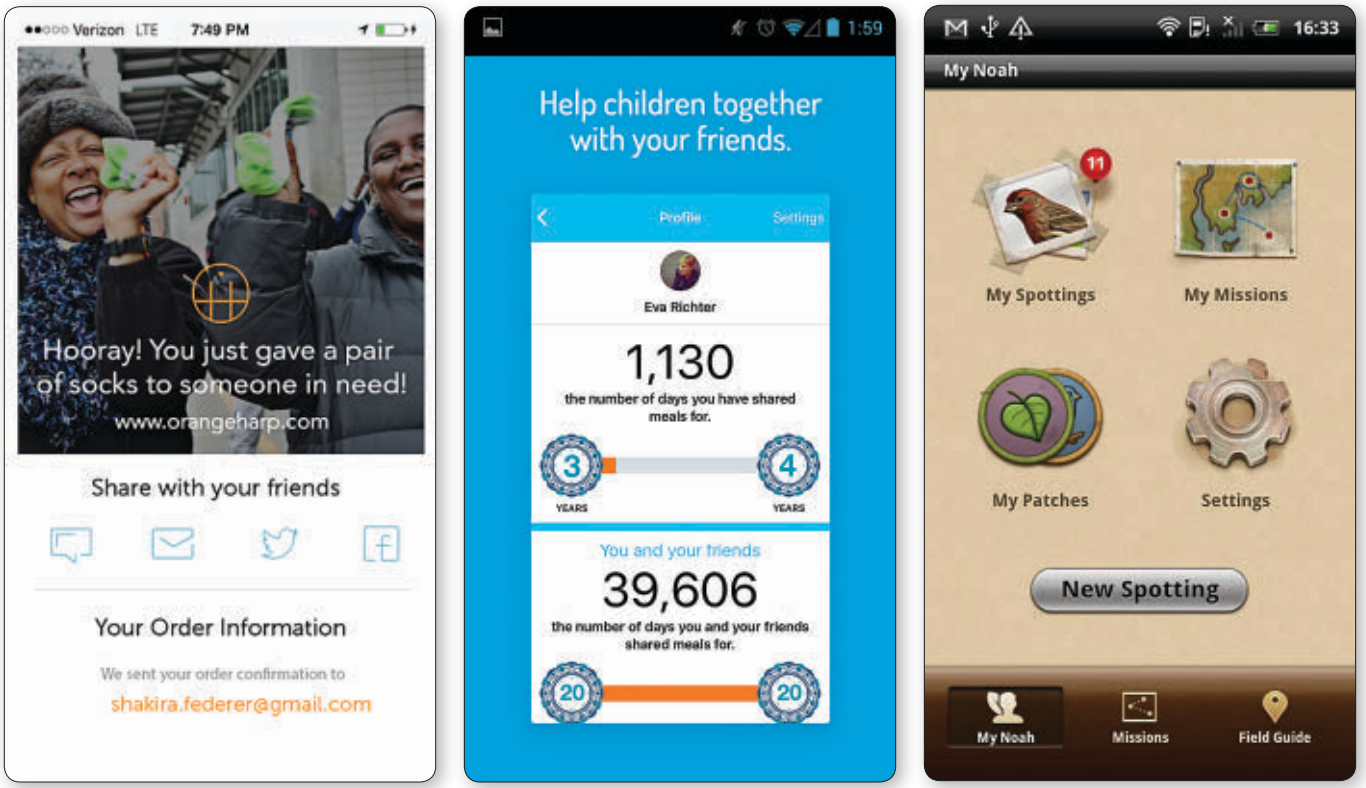
Global Impact

This is not just a movement for U.S.-based hipsters. The Lunchbox Fund, for example, provides daily meals to poor children in rural areas of South Africa. In 2013, South African cleric Desmond Tutu signed on as patron of the organization, which primarily solicited donations in his home country.

That same year, the Lunchbox Fund decided to expand its footprint in a cost-effective way, so it created the Feedie app to help it raise more money in other countries without hiring a huge canvassing force or running a global marketing campaign.

Here is how it works: Feedie users connect the app to their social networks. They then use the app to share photos of their meals in participating restaurants, and those restaurants make donations to The Lunchbox Fund to feed hungry children.

At press time, more than 12 million meals had been sponsored for children worldwide by Feedie.



SCREEN IMAGES COURTESY OF ORANGEHARP.COM, SHARETHEMEAL.ORG, AND PROJECTNOAH.ORG

Phone screens displaying apps that allow users to give to those in need; from left, Orange Harp, Share the Meal, and Project Noah.

ShareTheMeal is another app that does good in a cost-effective way by leveraging cross-border engagement. Developed by the United Nations World Food Program, the app makes it easy for a user to tap once and donate $0.50, enough to feed a child for an entire day in most of the world. Instead of costly outreach, users may download the app from anywhere in the world and donate. The U.N. claims this initiative's administrative costs are among the lowest in the non-profit world, with 90% of donations going directly to operations.

"So many people want to dedicate time and energy to charities that they feel passionate about, but there may be certain factors preventing them from doing so, like geographic barriers," says Sophie Barnett, the Lunchbox Fund's digital coordinator. "Social good apps, accessible from anywhere, at any time, have made doing good infinitely easier."

Social good apps succeed because they are a win-win-win for charitable organizations, users, and participating businesses. In the case of Feedie, restaurants pay $500 as a tax-deductible donation in anticipation of 2,000 photos of their meals being posted. They receive baked-in buzz as photos of their culinary creations are shared by users who feel good when they do good, and they garner media attention when Feedie is featured by media outlets like Mashable, The Huffington Post, and Time, marketing exposure for which many businesses would gladly pay. Thanks to Feedie's ingenuity, businesses pay to feed the hungry instead.

"We want to make taking a Feedie photo as ubiquitous as posting to Instagram and create a sustainable, scalable impact," says Barnett.

Doing Well While Doing Good

Feedie highlights an important truth about social good apps of any type: those that create the biggest impact are the ones that help others do well while doing good. That is exhibited in the business model of the MicroHero survey-taking app.

Users are asked to complete free surveys in MicroHero; each completed survey results in a donation to the user's favorite charity by companies that conduct market research, or businesses that have pressing questions they need answered. The result is a type of conscientious capitalism that gives businesses a direct incentive to support charitable causes. Currently, MicroHero works with over 200 charities, including the World Wildlife Fund and the American Red Cross.

Social good apps succeed because they are a win-win-win for charitable organizations, users, and participating businesses.

This model only works when you understand how consumers actually use smartphones; you cannot simply use an app notification to ask for a donation and call it a day, says MicroHero's de Brun, adding that apps must provide an experience just like any consumer product.

"Smartphone operating systems and app user interfaces are getting better at serving end users in natural, even delightful ways," he says. "If done well, social good actions can be very spontaneous and efficient from a phone."

In short, charitable giving cannot only become widespread on mobile devices; it can become second nature if done right, just like opening Facebook or checking Snapchat. There is an added financial benefit for consumers, too: you can claim many types of charitable donations as tax deductions. While the apps themselves do not offer much help with this (you still must claim a charitable donation on your own), they do make it easier to give in the first place.

De Brun believes social good apps like MicroHero are just getting started.

As user bases grow, social good app developers will find more fluid ways to connect in-app actions with actual monetary impact. Gamification will help: the more giving feels like play, the more engagement we will see from socially conscious users. Widespread adoption, says de Brun, will cause more corporate social responsibility programs to get in the game, and that is when things will start to get really interesting.

When social good apps appeal not just to individuals but entire corporations, their impact grows by orders of magnitude.

That is because the incentives for doing good and doing well are aligned. "Entrepreneurs are inventing business models that provide social good and provide for a profitable, sustainable business," de Brun says.

Other app makers agree. Nick Marino is director of Social Change at TangoTab, an app that donates a meal to local food charities each time a user checks in at a local restaurant. "I believe we have seen an overall increase in businesses working with causes," Marino says.

Restaurants like Starbucks, Nobu, and Maggiano's have signed up to work with TangoTab, because the best social good apps leverage consumer behavior to create real change, Marino explains.

"Apps like TangoTab are making it easier for people to make a difference by doing things they already do daily. People dine out all the time. We give people the opportunity to impact someone in need just by doing what they're already doing." C

Further Reading

Wigglesworth, V.,
(2015) SafeNight app comes to the aid of domestic violence victims in North Texas. *The Dallas Morning News.* http://www.dallasnews.com/news/domestic-violence/20151009-safenight-app-comes-to-the-aid-of-domestic-violence-victims-in-north-texas.ece

Godfrey, M,.
(2013) Feedie App Turns Food Photos Into Charitable Giving. *ABC News.* http://abcnews.go.com/Technology/feedie-app-turns-food-photos-charitable-giving/story?id=20060750

Chang, L,.
(2015) With the ShareTheMeal App, Your 50-Cent Donation Can Help End World Hunger. *Digital Trends.* http://www.digitaltrends.com/mobile/help-end-world-hunger-with-the-uns-sharethemeal-app/

Logan Kugler is a freelance technology writer based in Tampa, FL. He has written for over 60 major publications.

XRDS

Seeks Student Volunteers

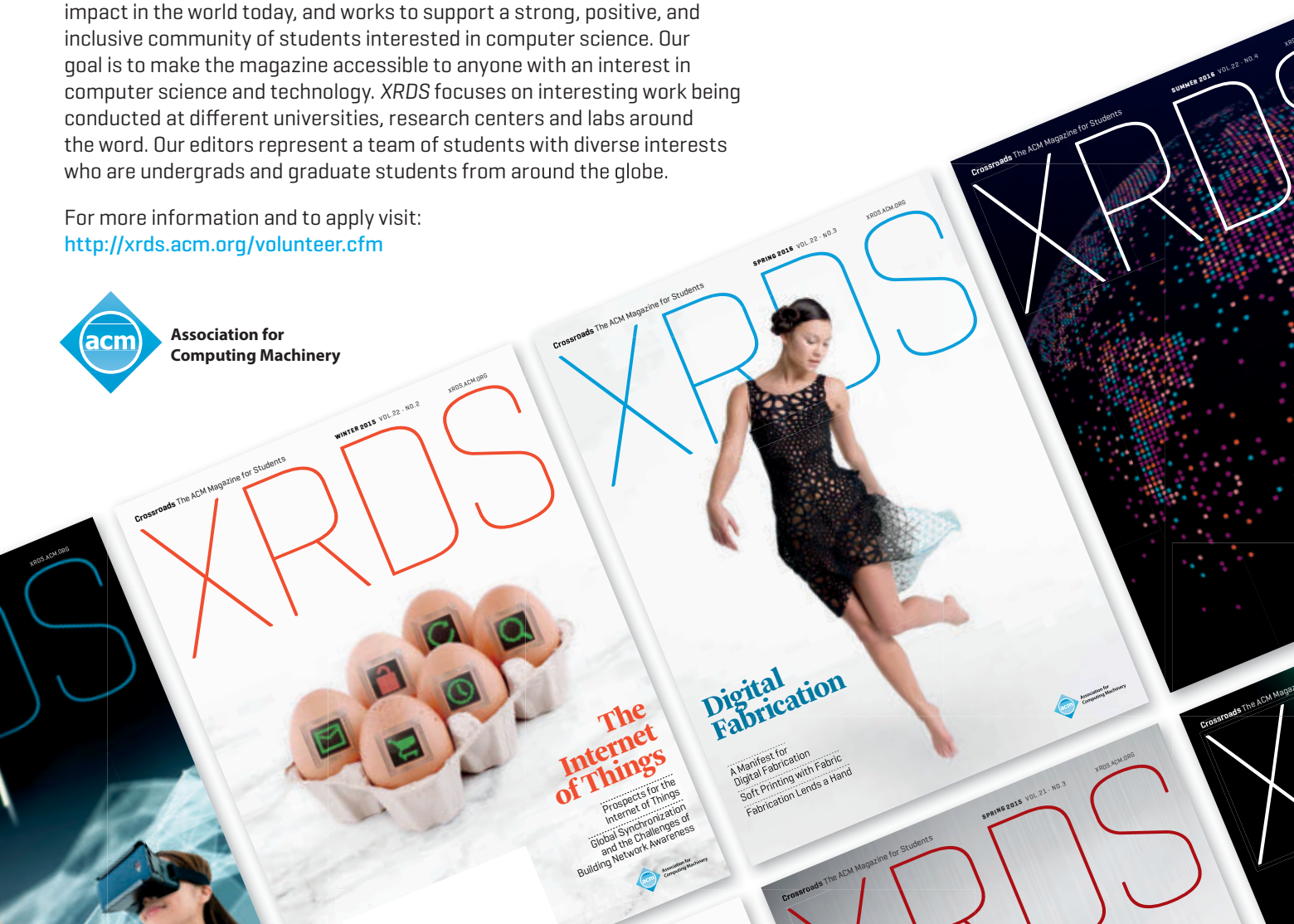
Are you a student who enjoys staying up to date with the latest tech innovations, and is looking to make an impact on the ACM community?

Editorial positions are now open for the following positions: social media editor, feature editor and department editor.

XRDS is a quarterly print magazine for students by students that examines cutting edge research in computer science, viewpoints on technology's impact in the world today, and works to support a strong, positive, and inclusive community of students interested in computer science. Our goal is to make the magazine accessible to anyone with an interest in computer science and technology. *XRDS* focuses on interesting work being conducted at different universities, research centers and labs around the word. Our editors represent a team of students with diverse interests who are undergrads and graduate students from around the globe.

For more information and to apply visit:
http://xrds.acm.org/volunteer.cfm

Association for Computing Machinery



DOI:10.1145/2955112

Paul Kocher

Privacy and Security
Computer Security Is Broken: Can Better Hardware Help Fix It?

Computer security problems have far exceeded the limits of the human brain. What can we do about it?

In 1967, the Silver Bridge collapsed into the Ohio River during rush hour. Instead of redundancy the bridge used high-strength steel. The failure of a single eyebar was catastrophic.[a] Today's computing devices resemble the Silver Bridge, but are much more complicated. They have billions of lines of code, logic gates, and other elements that must work perfectly. Otherwise, adversaries can compromise the system. The individual failure rates of many of these components are small, but aggregate complexity makes vulnerability statistically certain.

This is a scaling problem. Security-critical aspects of our computing platforms have seen exponential increases in complexity, rapidly overwhelming defensive improvements. The futility of the situation leads to illogical reasoning that structural engineers would never accept, such as claiming that obviously weak systems are "strong" simply because we are ignorant as to which specific elements will fail.

Security Building Blocks

To build strong security systems, we need reliable building blocks. Cryptographic algorithms are arguably the most important building block we have today. Well-designed algorithms can provide extraordinary strength against certain attacks. For example, Diffie-Hellman, RSA, and triple DES—known since the 1970s—continue to provide practical security today if sufficiently large keys are used.

To build strong security systems, we need better building blocks.

Protocols can greatly simplify security by removing reliance on communications channels, but in practice they have proven deceptively tricky. In 1996, I co-authored the SSL v3.0 protocol for Netscape, which became the basis for the TLS standard. Despite nearly 20 years of extensive analysis, researchers have identified new issues and corner cases related to SSL/TLS even relatively recently. Still, I believe we are reaching the point of having cryptographically sound protocols. Although breakthroughs in quantum computing or other attacks are conceivable, I am cautiously optimistic that current versions of the TLS standard (when conservative key sizes and configurations are chosen) will resist cryptanalysis over many decades.

Unfortunately, my optimism does not extend to actual implementations. Most computing devices running SSL/TLS are riddled with vulnerabilities that allow adversaries to make an end-run around the cryptography. For example, errant pointers in device drivers or bugs in CPU memory management units can

a "The Collapse of the Silver Bridge: NBS Determines Cause," 2009; https://1.usa.gov/21cRgUV

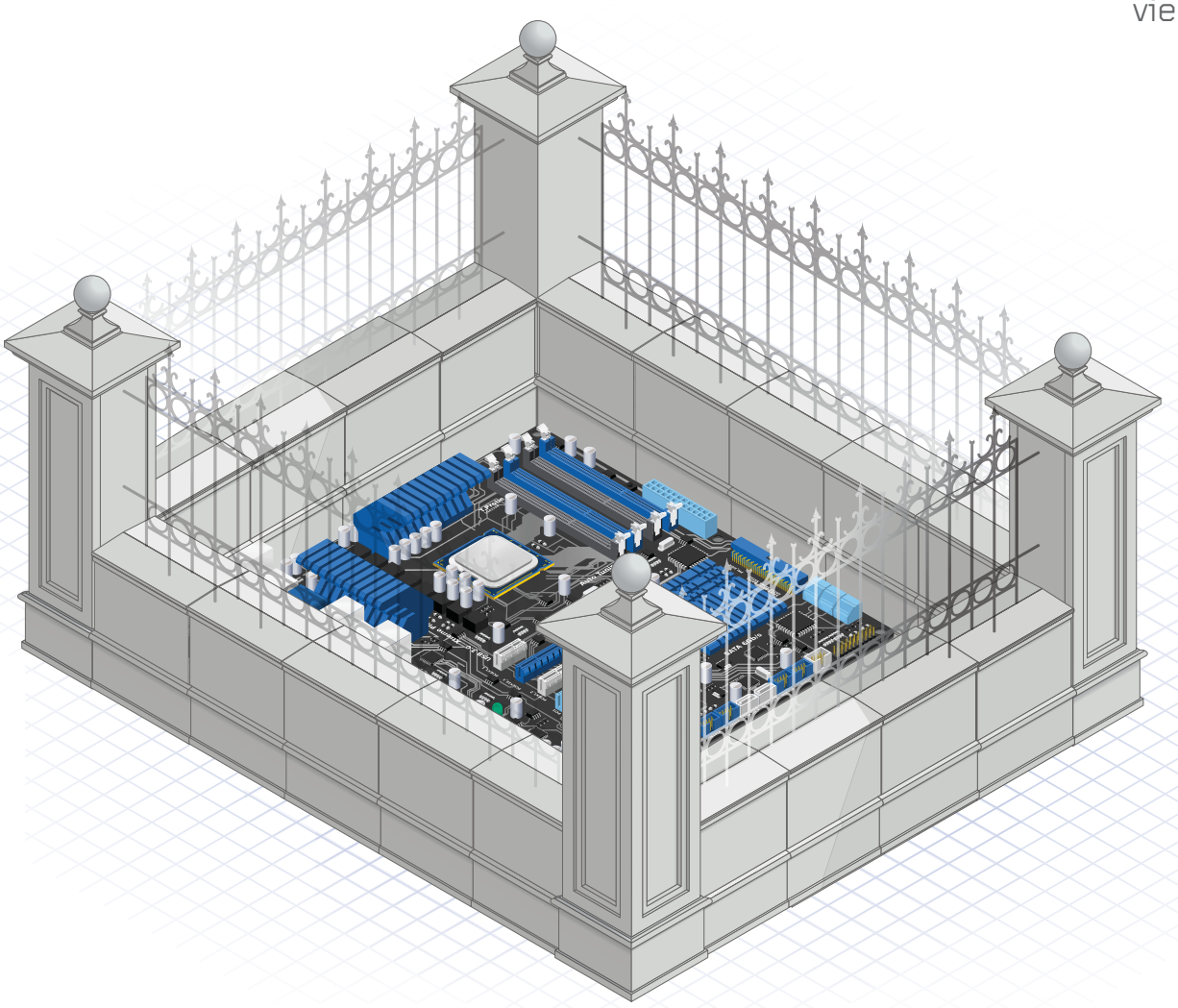


IMAGE BY ALICIA KUBISTA/ANDRIJ BORYS ASSOCIATES

destroy security for all software on a device. To make progress, we need another building block: simple, high-assurance hardware for secure computation.

Secure Computation in Hardware

Consider the problem of digitally signing messages while securing a private key. Meaningful security assurances are impossible for software implemented on a typical PC or smartphone due to reliance on complex and untrustworthy components, including the hardware, operating system, and so forth. Alternatively, if the computation is moved to isolated hardware, the private key's security depends only on the logic of a comparatively simple hardware block (see Figure 1). The amount of security-critical logic is reduced by many orders of magnitude, turning an unsolvable security problem into a reasonably well-bounded one.

In the 1990s, I started investigating secure hardware, assuming that simple mathematical operations like encryption and digital signing would be straightforward to secure. The problems I encountered were a lot more interesting and less intuitive than I expected.

I noticed small data-dependent correlations in timing measurements of devices' cryptographic operations. Cryptographic algorithms are extremely brittle; they are very difficult to break by analyzing binary input and output messages, but fail if attackers get any other information. The timing variations violated the algorithms' security model, and in practice allowed me to factor RSA keys and break other algorithms.[b]

I bought the cheapest analog oscilloscope from Fry's electronics and placed a resistor in the ground input of a chip doing cryptographic operations. The scope showed power consumption varying with the pattern of branches taken by the device's processor. I could easily identify the conditions for these branches—the bits of the secret key. Upgrading to a digital storage oscilloscope enabled far more advanced analysis methods. With my colleagues Joshua Jaffe and Benjamin Jun, I developed statistical techniques (Differential Power Analysis, or DPA) to solve for keys by leveraging tiny correlations in noisy power consumption or RF measurements.[c]

Side channels weren't the only issue. For example, scan chains and other test modes can be abused by attackers. Researchers and pay TV pirates independently discovered that glitches and other computation errors can be devastating for security.[d]

Fortunately, practical and effective solutions have been found and implemented to these issues. For example, nearly 10 billion chips are made annually with DPA countermeasures.

b P. Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems," 1996; https://bit.ly/25S86vt

c P. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," 1999; https://bit.ly/1XLZhSZ

d D. Boneh, R. DeMillo, and R. Lipton, "On the Importance of Checking Cryptographic Protocols for Faults," 1997; https://stanford.io/1PPPFnj

Figure 1. A simple fixed-function secure computation block.

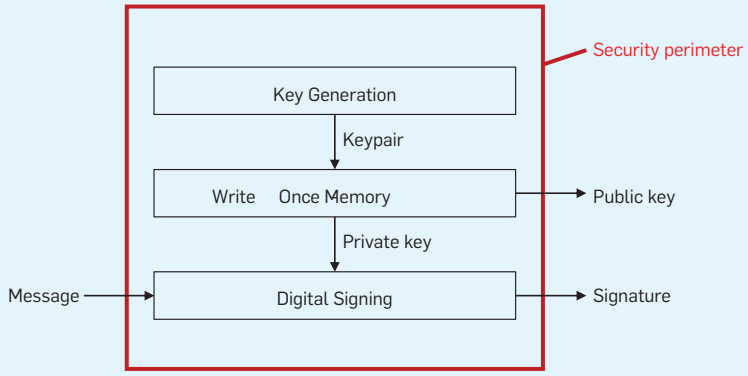
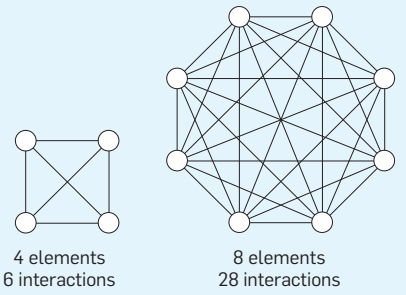


Figure 2. Vulnerability risks increase with the number of potential interactions between system elements.



Although there is a possibility that unexpected new categories of attack may be discovered, based on what we know, a well-designed chip can be robust against non-invasive attacks. Strategies for addressing invasive attacks have also improved greatly, although still generally assume some degree of security through obscurity.

Adding Secure Computation to Legacy Architectures

Today's computing architectures are too entrenched to abandon, yet too complex to secure. It is practical, however, to add extra hardware where critical operations can be isolated. Actual efforts to do this in practice vary wildly in cost, programming models, features, and level of security assurance.

Early attempts used standalone security chips, such as SIM cards in mobile devices, TPMs in PCs, and conditional access cards in pay TV systems. These were limited to single-purpose applications that could bear the cost—typically a dollar or more. The security chip's electrical interface was also a security risk, for example allowing pay TV pirates to steal video decryption keys for redistribution.

Another strategy is to add security modes to existing designs. Because the existing processor and other logic are reused, these approaches add almost no die area. Unfortunately, this reuse brings significant security risks due to bugs in the shared logic and separation mechanisms. Intel's Software Guard Extensions (SGX)[e] leave almost all of the (very complex[f]) processor in the security perimeter, and do not even appear to mitigate side channel or glitch attacks. Trusted Execution Environments (TEEs) typically use ARM's TrustZone CPU mode to try to isolate an independent "trusted" operating system, but security dependencies include the CPU(s), the chip's test/debug modes, the memory subsystem/RAM, the TEE operating system, and other high-privilege software.

The approach I find most compelling is to integrate security blocks onto large multi-function chips. These cores can create an intra-chip security perimeter that does not trust the RAM, legacy processors, operating system, or other logic. In addition to providing much better security integration than separate chips, on-die cores cost 1–2 orders of magnitude less. Examples of on-chip security hardware include Apple's Secure Enclave, AMD's Secure Processor, and Rambus's CryptoManager Cores. Depending on the application, a security core may offload specific functions like authentication, or can be programmable. Over time, these secure domains can improve and evolve to take on a growing range of security-sensitive operations.

Better hardware foundations can enable a new evolutionary process that is essential for the technology industry's future.

Limits of Human Comprehension

Security building blocks must be simple enough for people to comprehend their intended security properties. With remarkable regularity, teams working on data security dramatically overestimate what can be implemented safely. I set a requirement when working on the design of SSL v3.0 that a technically proficient person could read and understand the protocol in one day. Despite this, multiple reviewers and I missed several important but subtle design problems, some of which were not found until many years later.

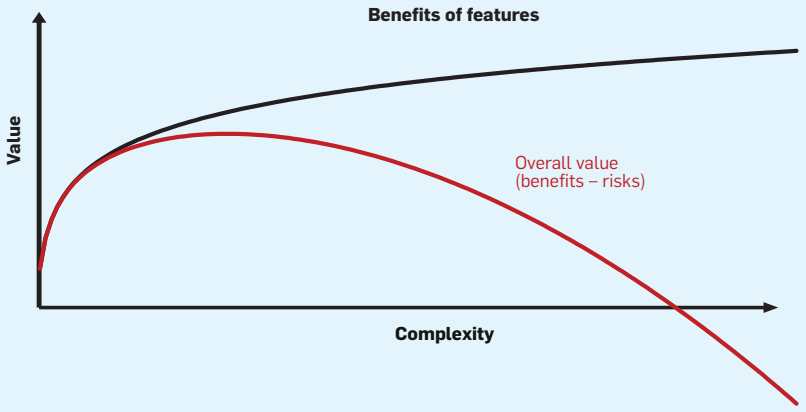
Vulnerability risks grow with the number of potential interactions between elements in the system. If interactions are not constrained, risks scale as the square of the number of elements (see Figure 2).

Although secure hardware blocks may appear to be simple, they are still challenging to verify. Formal methods, security proofs, and static analysis tools can help to some extent by augmenting our human brains. Still, there are gaps between these methods and the messiness of the real world. As a result, these ap-

e M. Hoekstra, "Intel SGX for Dummies," 2013; https://intel.ly/1eQpY4P

f J. Rutkowska, "Intel x86 Considered Harmful," 2015; https://bit.ly/1ObbBaA

Figure 3. Increased complexity increases risk.



proaches need to be combined with careful efforts to keep complexity under control. For example, expanding a system from 8 to 11 elements approximately doubles the number of potential interactions. A tool that eliminated 50% of defects would be extraordinary, yet this modest increase in complexity could exhaust its benefits. Nevertheless, these approaches can help us extend our abilities as we work to create new building blocks.

Security and the Technology Industry's Future

The benefits of feature enhancements to existing products have diminishing returns. The most important capabilities get implemented first. Doubling the lines of code in a word processor will not make the program twice as useful. A smartphone with two CPUs is only a little better than a phone with one.

From a security perspective, extra features can create risks that scale faster than the increase in complexity (see Figure 3). As a result, the technology industry faces a troubling curve: as complexity increases, the benefits from added features are undermined and ultimately overwhelmed by risk. When more advanced products become less valuable, innovation stalls.

Many applications are near or already beyond the point where value begins to decline without new approaches to security. Current efforts to build the 'Internet of Things' or 'Smart Cities' using conventional hardware and operating systems will produce connected systems that are plagued with hidden vulnerabilities. The costs of coping with these weaknesses, and the serious failures that will occur anyway, can easily overwhelm the benefits.

Secure compute building blocks do not eliminate the link between complexity and risk, but can fundamentally change the risk calculus. Critical functions can scale independently from each other and from the rest of the system. Each security-relevant use case is exposed to dramatically less overall complexity and can be optimized separately on the value/complexity curve.

Although Moore's Law is slowing, transistor costs will continue to fall. If hardware budgets for security stay constant, the number of security blocks that can be added to each chip will increase exponentially. These blocks can be tailored for different use cases. They can also be organized with redundancy to avoid single points of failure like the one that doomed the Silver Bridge.

Better hardware foundations can enable a new evolutionary process that is essential for the technology industry's future. Problems that are unsolvable today due to unconstrained interconnectedness can be isolated. Over time, secure domains can improve and evolve, addressing a growing range of security-sensitive needs. Eventually, claims of security may even be based on a realistic understanding of what humans can successfully implement. C

Paul Kocher (paul@paulkocher.com) is President and Chief Scientist in the Cryptography Research division of Rambus in Sunnyvale, CA.

Calendar of Events

August 4– 5
APSys '16: 7th ACM SIGOPS Asia-Pacific Workshop on Systems,
Hong Kong, China,
Sponsored: ACM/SIG,
Contact: Heming Cui,
Email: heming@cs.hku.hk

August 13–17
KDD '16: The 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining,
San Francisco, CA,
Co-Sponsored: ACM/SIG,
Contact: Mohak Shah,
Email: mohak.shah@us.bosch.com

August 18–21
ASONAM '16: Advances in Social Networks Analysis and Mining 2016,
Davis, CA,
Contact: Jon Rokne,
Email: rokne@ucalgary.ca

August 22–26
SIGCOMM '16: ACM SIGCOMM 2016 Conference,
Salvador, Brazil,
Sponsored: ACM/SIG,
Contact: Jon Crowcroft,
Email: jac22@cl.cam.ac.uk

August 29–Sept 3
SBCCI '16: 29th Symposium on Integrated Circuits and Systems Design,
Belo Horizonte, Brazil,
Contact: Davies William de Lima Monteiro,
Email: davies@ufmg.br

September

September 3–7
ASE '16 ACM/IEEE International Conference on Automated Software Engineering,
Singapore,
Contact: David Lo,
Email: davidlo@smu.edu.sg

September 6–9
MobileHCI '16: 18th International Conference on Human-Computer Interaction with Mobile Devices and Services,
Florence, Italy,
Sponsored: ACM/SIG,
Contact: Fabio Paterno,
Email: fabio.paterno@isti.cnr.it

DOI:10.1145/2955114

Yasmin B. Kafai

Education

From Computational Thinking to Computational Participation in K–12 Education

Seeking to reframe computational thinking as computational participation.

COMPUTATIONAL THINKING has become a battle cry for coding in K–12 education. It is echoed in statewide efforts to develop standards, in changes to teacher certification and graduation requirements, and in new curriculum designs.[1] The annual Hour of Code has introduced millions of kids to coding inspired by Apple cofounder Steve Jobs who said, "everyone should learn how to program a computer because it teaches you how to think." Computational thinking has garnered much attention but people seldom recognize that the goal is to bring programming *back* into the classroom.

In the 1980s many schools featured Basic, Logo, or Pascal programming computer labs. Students typically received weekly introductory programming instruction.[6] These exercises were often of limited complexity, disconnected from classroom work, and lacking in relevance. They did not deliver on promises. By the mid-1990s most schools had turned away from programming. Pre-assembled multimedia packages burned onto glossy CD-ROMs took over. Toiling over syntax typos and debugging problems were no longer classroom activities.

Computer science is making a comeback in schools. We should not repeat earlier mistakes, but leverage what we have learned.[5] Why are students interested in programming? Under what circumstances do they do it, and how?[2] Computational thinking and programming are social, creative practices. They offer a context for making applications of significance for others, communities in which design sharing and collaboration with others are paramount. Computational *thinking* should be reframed as computational *participation*.

Students in a Makey Makey workshop conducted by volunteers from Robogals Wellesley.

PHOTO COURTESY OF ROBOGALS WELLESLEY

Computational Participation

This idea expands on Jeannette Wing's original definition of computational thinking.[7] *Computational participation* involves solving problems, designing systems, and understanding human behavior in the context of computing. It allows for participation in digital activities. Many kids use code outside of school to create and share. Youth-generated websites have appeared to make and share programmable media online. These sites include video games, interactive art projects, and digital stories. They are inherently do-it-yourself (DIY), encouraging youth programming as an effective way to create and share online,

and connect with each other, unlike learned disciplines such as algebra or chemistry. Through individual endeavor mixed with group feedback and collaboration the DIY ethos opens up three new pathways for engaging youth.

From building code to creating shareable applications. Programming that prizes coding accuracy and efficiency as signifiers of success is boring. To learn programming for the sake of programming goes nowhere for children unless they can put those skills to use in a meaningful way. Today children program to create applications like video games or interactive stories as part of a larger learning community.[3] They are attracted to the possibility of creating something real and tangible that can be shared with others. Programming is not an abstract discipline, but a way to "make" and "be" in the digital world.

From tools to communities. Coding was once a solitary, tool-based activity. Now it is becoming a shared social practice. Participation spurred by open software environments and mutual enthusiasm shifts attention from programming tools to designing and supporting communities of learners. The past decade has brought many admirable introductory programming languages to make coding more intuitive and personal. Developers and educators realize that tools alone are not enough. Audiences are needed, and a critical mass of like-minded creators. Scratch, Alice, and similar tools have online communities of millions of young users. Children can work and share programs on a single website. This tacitly highlights the community of practice that has become a key for learning to code.

From "from scratch" creation to "remixing." These new, networked communities focus on *remixing*. Students once created programs from scratch to demonstrate competency. Now they pursue seamless integration via remixing as the new social norm, in the spirit of the open source movement. Sharing one's code encourages others to sample creations, adjust them, and add to them. Such openness heightens potential for innovation across the board. Young users embrace sampling and sharing more freely, challenging the traditional top-down paradigm.

These three shifts signal a social turn in K–12 computing. They move from a predominantly individualistic view to greater focus on underlying social and cultural dimensions of programming. We should rethink what and how students learn to become full participants in networked communities.

How do we facilitate broader and deeper participation in the design of the programming activities, tools, and practices?

Broadening and Deepening Computational Participation

It is not possible to addresses all of the difficulties of implementing computational participation by placing students in groups, having them program applications, and encouraging them to remix code. Computational participation will present new challenges in bringing programming back into schools. How do we facilitate broader and deeper participation in the design of the programming activities, tools, and practices?

Computational thinking is a social practice. We must broaden access to communities of programming.[4] Children are not "digital natives" who naturally migrate online. Establishing membership in the programming community is not easy. Groups with powerful learning cultures are often exclusive cultures. Students need strategies to cope with the vulnerability of sharing one's work for others to comment on and remix.

In addition, students need a more expansive menu of computing activities, tools, and materials. Designing authentic applications is an important step in the right direction, but games, stories, and robotics are not the only applications for achieving this goal. We need different materials to expand students' perspectives and perceptions of computing.

Broadening computational participation gets students into the clubhouse. The next challenge is to help them develop fluency that permits them to engage deeply, making their participation meaningful and enriching. These levels of computational participation are still rare. To learn to code students must learn the technicalities of programming language and common algorithms, *and* the social practices of programming communities.

Conclusion

Computational participation provides new direction for programming in K–12 education. It moves us beyond tools and code to community and context. It equips designers, educators, and researchers to broaden and deepen computational thinking on a larger scale than previously. Users of digital technologies for functional, political, and personal purposes need a basic understanding of computing. Students must understand interfaces, technologies, and systems that they encounter daily. This will empower them and provides them with the tools to examine and question design decisions they encounter. Computing for communicating and interacting with others builds relationships. Education activist Paulo Freire once said that "reading the word is reading the world." He was right. Today, reading code is about reading the world. It is needed to understand, change, and remake the digital world in which we live. C

References

1. Grover, S. and Pea, R. Computational thinking in K–12: A review of the state of the field. *Educational Researcher 42*, 2 (2013), 59–69.
2. Kafai, Y.B. and Burke, Q. *Connected Code: Why Children Need to Learn Programming*. MIT Press, Cambridge, 2014.
3. Kafai, Y.B. and Burke, Q. Constructionist gaming: Understanding the benefits of making games for learning. *Educational Psychologist 50*, 4 (2015), 313–334.
4. Margolis, J. Estrella, E., Goode, G., Holme, J. J. and Nao, K. *Stuck in the Shallow End: Race, Education, and Computing*. MIT Press, Cambridge, 2008.
5. Palumbo, D.B. Programming language/problem-solving research: A review of relevant issues. *Review of Educational Research 60*, 1 (1990), 65–89.
6. Papert, S. Mindstorms: Children, Computers, and Powerful Ideas. Basic Books, New York, 1980.
7. Wing, J.M. Computational thinking. *Commun. ACM 49*, 3 (Mar. 2006), 33–35.

Yasmin B. Kafai (kafai@gse.upenn.edu) is a Professor and Chair of the Teaching, Learning, and Leadership Division in the Graduate School of Education at the University of Pennsylvania.

DOI:10.1145/2955116

George V. Neville-Neil

Article development led by acmqueue
queue.acm.org

Kode Vicious
Chilling the Messenger

Keeping ego out of software-design review.

Dear KV,

I was recently hired as a mid-level Web developer working on version 2 of a highly successful but outdated Web application. It will be implemented with ASP.Net WebAPI. Our architect designed a layered architecture, roughly like Web Service > Data Service > Data Access. He noted that data service should be agnostic to Entity Framework ORM (object-relational mapping), and it should use unit-of-work and repository patterns. I guess my problem sort of started there.

Our lead developer has created a solution to implement the architecture, but the implementation does not apply the unit-of-work and repository patterns correctly. Worse, the code is really difficult to understand and it does not actually fit the architecture. So I see a lot of red flags coming up with this implementation. It took me almost an entire weekend to work through the code, and there are still gaps in my understanding.

This week our first sprint starts, and I feel a responsibility to speak up and try to address this issue. I know that I will face a lot of resistance, just based on the fact that the lead developer wrote that code and understands it more than the alternatives. He may not see the issue that I will try to convey. I need to convince him and the rest of the team that the code needs to be refactored or reworked. I feel apprehensive, because I am like the new kid on the block trying to change the game. I also don't want to be perceived as Mr. Know-It-All, even though I might be a little more opinionated than I should be sometimes.

IMAGE BY MARYNA PLESHKUN

My question is, how can I convince the team that there is a real problem with the implementation without offending anyone?

~Opinionated

Dear ~Opinionated,

Let me work backward through your letter from the end. You are asking me, Kode Vicious, how to point out problems without offending anyone? Have you read any of my previous columns? Let's just start out with the KV ground rules: it's only the law and other deleterious side effects that keep me on the "right" side of violence in some

meetings. I'd like to think a jury of my peers would acquit me should I eventually cross to the wrong side, but I don't want to stake my freedom on that. I will try my best to give you solutions that do not land you in jail, but I will not guarantee them not to offend.

Trying to correct someone who has just done a lot of work, even if, ultimately, that work is not the right work, is a daunting task. The person in question no doubt believes that he has worked very hard to produce something of value to the rest of the team, and walking in and spitting on it, literally or metaphorically, probably crosses your "offense" line—at least I think it does. I'm a bit surprised that since this is the first sprint and there is already so much code written, shouldn't the software have shown up after the sprints established what was needed, who the stakeholders were, and so forth? Or was this a piece of previously existing code that was being brought in to solve a new problem? It probably doesn't matter, because the crux of your letter is the fact that you and your team do not sufficiently understand the software in question to be comfortable with fielding it.

In order to become more comfortable with the system, there are two things to call for: a design review and a code review. These are not actually the same things, and KV has already covered how to conduct a code review (see my October 2009 *Communications* column Kode Reviews 101). Let's talk now about a design review.

A software design review is intended to answer a basic set of questions:

1. How does the design take inputs and turn them into outputs?
2. What are the major components that make up the system?
3. How do the components work together to achieve the goals set out by the design?

That all sounds simple, but the devil is in the level of the details. Many software developers and systems architects would prefer that everyone but themselves see the systems they have built as black boxes, where data goes in and other data comes out, no questions asked. You clearly do not have the necessary level of trust with the software you're working with to allow the lead developer to get away with that, so you should call for a design review where you take the lid off the box and poke around at the parts inside. In fact, questions 2 and 3 are going to be your main tools for figuring out what the software does and whether or not it is suitable for the task.

The one thing *not* to do in a design review is to turn it into a code review.

When I have to interview people for jobs, I always ask them questions about systems they have worked on while we draw out the block diagram on a whiteboard: What are the major components? How does component A talk to component B? What happens if C fails? I'm trying to transfer their mental images of their software into my own mind, of course without either going mad or having a nasty flashback. Some pieces of software are best left outside your mind, but hopefully that's not going to be the case with the system you're working with.

Remember that every box this person draws can be opened if you think you're not getting sufficient detail. Much like the ancient game show, "Let's Make a Deal," it is always OK for you to ask, "What's behind door number 1, Monty?" Of course, you might find that it's a goat, but hopefully you find that it's a working set of components that are understandable to you and the team.

The one thing *not* to do in a design review is turn it into a code review. You are definitely not interested in the internals of any of the algorithms, at least not yet. The only code you might want to look at are the APIs that glue the components together, but even these are best left abstract, so that the amount of detail does not overwhelm you. Remember that the goal is always to get the big picture rather than the fine details, at least in a design review.

Returning to the question of offense, I have found only one legal way to avoid giving offense, and that is always to phrase things as questions. Often called the Socratic method, this can be a good way to get people to explain to you, and often to themselves, what they think they are doing. The Socratic method can be applied in an annoyingly pedantic way, but since you're trying not to give offense, I suggest you play by a few useful rules. First, do not hammer the person with a relentless list of questions right off. Remember that you are trying to explore the design space in a collaborative way; this is not an interrogation. Second, leave spaces for the people you're working with to think. A pause doesn't mean they don't know; in fact, it might be that they're trying to adjust their mental model of the system in a way that will be beneficial to everyone when the review is done. Lastly, try to vary the questions you ask and the words you use. No one wants to be subjected to a lot of, "And then what happens?"

Finally, I find that when I'm in a design review and about to do something that might give offense, such as throwing a chair or a whiteboard marker, I try to do something less obvious. My personal style is to take off my glasses, put them on the table and speak in a very calm voice. That usually doesn't offend, but it does get people's attention, which leads them to concentrate harder on working to understand the problem we're all trying to solve.

KV

Related articles on queue.acm.org

The Code Delusion
Stan Kelly-Bootle
http://queue.acm.org/detail.cfm?id=1317411

Verification of Safety-critical Software
B. Scott Andersen and George Romanski
http://queue.acm.org/detail.cfm?id=2024356

Lazarus Code
George Neville-Neil
http://queue.acm.org/detail.cfm?id=2773214

George V. Neville-Neil (kv@acm.org) is the proprietor of Neville-Neil Consulting and co-chair of the *ACM Queue* editorial board. He works on networking and operating systems code for fun and profit, teaches courses on various programming-related subjects, and encourages your comments, quips, and code snips pertaining to his *Communications* column.

DOI:10.1145/2955110

Ben Shneiderman

Viewpoint
Teamwork in Computing Research

Considering the benefits and downsides of collaborative research.

Teamwork has long been a part of computing research, but now advanced technologies and widespread proficiency with collaboration technologies are creating new opportunities. The capacity to share data, computing resources, and research instruments has been growing steadily, just as predicted when Bill Wulf coined the term "collaboratories" a quarter of a century ago.[2,15,16]

Teamwork has become the overwhelmingly dominant form of research, so rewarding effective teams, teaching our students how to collaborate, and supporting research on what works and what doesn't have taken on new importance.

The Case For and Against Teamwork

In recent years, the tools for locating relevant documents, finding team members with special skills, coordinating schedules, and refining reports collaboratively has grown substantially.[7] In addition to these technology advances, the willingness and fluency with which young researchers appear to use video and audio conferencing, curated datasets, shared document editors, task managers, and other collaboration tools grows steadily.

Another driving force for teamwork and larger group collaborations is what I see as the increased ambition of team members and the growing expectations from research leaders. Teamwork brings more than larger capacity for work; it opens new possibilities when different disciplines, research methods, and personalities are fruitfully combined.[6] In short, research teams of two to 10 members and larger groups of hundreds of researchers, can accomplish much more and conduct higher quality work than they could just two decades ago when the Internet was a novelty.[10]

The growth of research teams was well documented in a series of papers by Wuchty, Jones, and Uzzi.[5,13,14] They reported that from the 1950s to 2000 the average number of journal paper co-authors in science and engineering grew from 1.9 to 3.5. They also showed that the impact as measured by journal citation counts increased as the number of authors grew. Furthermore, the benefit of teamwork grew over time. In 1955, team papers attracted 1.7 times as many citations as solo-authored papers, but by 2000 the advantage grew to 2.1 times as many citations, suggesting that technologies and teamwork skills had enabled teams to be more effective than they were in the past.

A study of the papers for the ACM SIGKDD 2014 Conference, by program chairs Jure Leskovec and Wei Wang, added evidence of the benefits of teamwork. The reviewer ratings of the 1,036 submitted papers increased steadily for papers with up to five co-authors, then remained level. Reviewer ratings may be imperfect, but this bit of evidence seems potent, especially since this conference had an impressively rigorous acceptance rate of approximately 14%. Another outcome was tied to the ratings for papers. Those papers whose authors included a mix of academics and business practitioners had statistically significantly higher ratings than papers whose authors were either all academics or all business practitioners. This added to the evidence that diversity in teams is a catalyst for high quality.[8]

However, teamwork has downsides, requiring extra coordination among collaborators, learning new disciplines, adjusting to fresh research methods, and accommodating different personalities.[4] These downsides mean that those who engage in teamwork will need to learn how to do it effectively, so as to attain the full benefits.

Doing Teamwork Right

The interest in research teams and larger groups has now accelerated with the publication of the National Academies report *Enhancing the Effectiveness of Team Science*.[3] This report adds recent evidence that teams and larger groups are a growing phenomenon in science and engineering research, where multiple authorship has risen to 90% of all papers in 2013.

The report makes another useful contribution by characterizing seven dimensions that challenge today's research teams (see the accompanying table). Teams on the left side of the range are easier to manage, while teams on the right side of the range are more difficult to manage, suggesting that these deserve more study.[3] The report offers suggestions of how to improve team processes and calls for increased research on teams and larger groups.

Teams and larger groups of academics and practitioners seem likely to be more

effective in choosing meaningful problems, forming successful research plans, and in testing hypotheses in living laboratories at scale. Teaming between applied and basic researchers is likely to be a growing trend, as indicated by a recent National Science Foundation program announcement, Algorithms in the Field:[11] "Algorithms in the Field encourages closer collaboration between two groups of researchers: (i) theoretical computer science researchers, who focus on the design and analysis of provably efficient and provably accurate algorithms for various computational models; and (ii) applied researchers including a combination of systems and domain experts."

Teamwork between academics and practitioners can have strong benefits, as does multidisciplinary collaboration within academic communities. A clear testimonial for joint research bridging computer science and other disciplines comes from David Patterson's review of his 35-plus years running research labs on computer systems: "The psychological support of others also increases the collective courage of a group. Multidisciplinary teams, which increasingly involve disciplines outside computer science, have greater opportunity if they are willing to take chances that individuals and companies will not."[9]

Patterson concludes with his vision of the growth of multidisciplinary teams: "Whereas early computing problems were more likely to be solved by a single investigator within a single discipline, I believe the fraction of computing problems requiring multidisciplinary teams will increase."

There is always room for solitary researchers who wish to pursue their own projects. There are substantial advantages to working alone, but those who learn team skills are more likely to become part of breakthroughs than those who go it alone.

A near-term impediment to teamwork is the difficulty that researchers expect to have when facing hiring, tenure, and promotion committees, who are perceived as having trouble in assessing individuals who contribute to teams. Even team members who have participated in many award-winning papers fear they will find it difficult to convince review committees of their contributions. Being a first author helps gain recognition, as does documenting the role of each team member in the acknowledgments section. Writing a single-author paper, when this is warranted, may also help in many disciplines.

Recommendations

In light of the growing interest in teamwork, appointment, promotion, and tenure committees would do well to update their methods for documenting and assessing teamwork, so as to encourage and reward effective team participation and leadership. One example to follow is the University of Southern California, which has developed guidelines to emphasize a variety of forms of collaborative scholarship and to introduce attribution standards for contributions to larger projects.[12]

A second step, for computing educators, would be to increase teamwork training and experiences, so as to raise the quality of students' work. Team projects in undergraduate and graduate courses would train students in using collaboration tools and nurture their communication skills for future professional or research jobs. The National Academies report makes many recommendations that need to be tailored to fit local computing cultures and the seven dimensions on which today's research teams differ in complexity.

A third step, for government agencies, would be to increase funding to study computing research teams so as to enable leaders to form and manage successful teams. Teamwork is difficult since it requires different skills than working alone, but the potential for greater impact makes teamwork attractive. A strong research agenda would include applied and basic components to understand which incentives and rewards best amplify success within the seven dimensions of the National Academies report. C

Seven dimensions that challenge today's research teams.

Dimension	Range	
Diversity of team or group membership	Homogeneous	Heterogeneous
Disciplinary integration	Unidisciplinary	Transdisciplinary
Team or group size	Small (2)	Mega (1000s)
Goal alignment across teams	Aligned	Divergent or Misaligned
Permeable team and organizational boundaries	Stable	Fluid
Proximity of team or group members	Co-located	Globally distributed
Task interdependence	Low	High

References

1. ACM 2014 Conference on Knowledge Discovery and Data Mining (ACM-KDD); www.kdd2014.org
2. Cerf, V.G. et al. *National Collaboratories: Applying Information Technologies for Scientific Research.* National Academy Press, Washington, D.C. (1993).
3. Cooke, N.J. and Hilton, M.L., Eds. *Enhancing the Effectiveness of Team Science.* National Academies Press, Washington, D.C. (2015); http://www.nap.edu/19007
4. Cummings, J. and Kiesler, S. Collaborative research across disciplinary and organizational boundaries. *Social Studies of Science 35*, 5 (2005), 703–722.
5. Jones, B.F., Wuchty, S., and Uzzi, B. Multi-University research teams: Shifting impact, geography, and stratification in science. *Science 322*, 5905 (2008), 1259–1262; DOI:10.1126/science.1158357
6. Nelson, B. The data on diversity. *Commun. ACM 57*, 11 (Nov. 2014), 86–95.
7. Olson, J.S. and Olson, G.M. *Working Together Apart: Collaboration over the Internet.* Morgan & Claypool Publishers, 2013.
8. Page, S.E. *The Difference: How the Power of Diversity Creates Better Groups, Firms, Schools, and Societies.* Princeton University Press, NJ, 2007.
9. Patterson, D. How to build a bad research center. *Commun. ACM 57*, 3 (Mar. 2014), 33–36.
10. Spector, A., Norvig, P., and Petrov, S. Google's hybrid approach to research, *Commun. ACM 55*, 7 (July 2012), 34–37.
11. U.S. National Science Foundation Program on Algorithms in the Field (AitF); http://www.nsf.gov/pubs/2015/nsf15515/nsf15515.htm
12. University of Southern California, Guidelines for Assigning Authorship and for Attributing Contributions to Research Products and Creative Works (Sept. 16, 2011); http://www.usc.edu/academe/acsen/Documents/senate%20news/URC_on_Authorship_and_Attribution_20110916.pdf
13. Wuchty, S., Jones, B. F., and Uzzi, B., The increasing dominance of teams in production of knowledge. *Science 316*, 5827 (2007a), 1036–1039. DOI:10.1126/science.1136099
14. Wuchty, S., Jones, B. F., and Uzzi, B., Why do team-authored papers get cited more? Response. *Science 317*, 5844, (2007b), 1497–1498.
15. Wulf, W.A. The collaboratory opportunity. *Science 261*, 5123 (1993), 854–855; DOI: 10.1126/science.8346438
16. Wulf, W.A. The National Collaboratory—A White Paper. In Towards a National Collaboratory, the unpublished report of a workshop held at Rockefeller University, March 17–18, 1989 (Joshua Lederberg and Keith Uncapher, co-chairs).

Ben Shneiderman (ben@cs.umd.edu) is a Distinguished University Professor of computer science at the University of Maryland, a Member of the National Academy of Engineering, and the author of *The New ABCs of Research: Achieving Breakthrough Collaborations* (Oxford University Press, April 2016).

Thanks to the anonymous reviewers and readers of early drafts, including Nancy Cooke, Gerhard Fischer, Kara Hall, Margaret Hilton, Judy Olson, Scott Page, Dave Patterson, and Jennifer Preece.

DOI:10.1145/2909480

Article development led by acmqueue
queue.acm.org

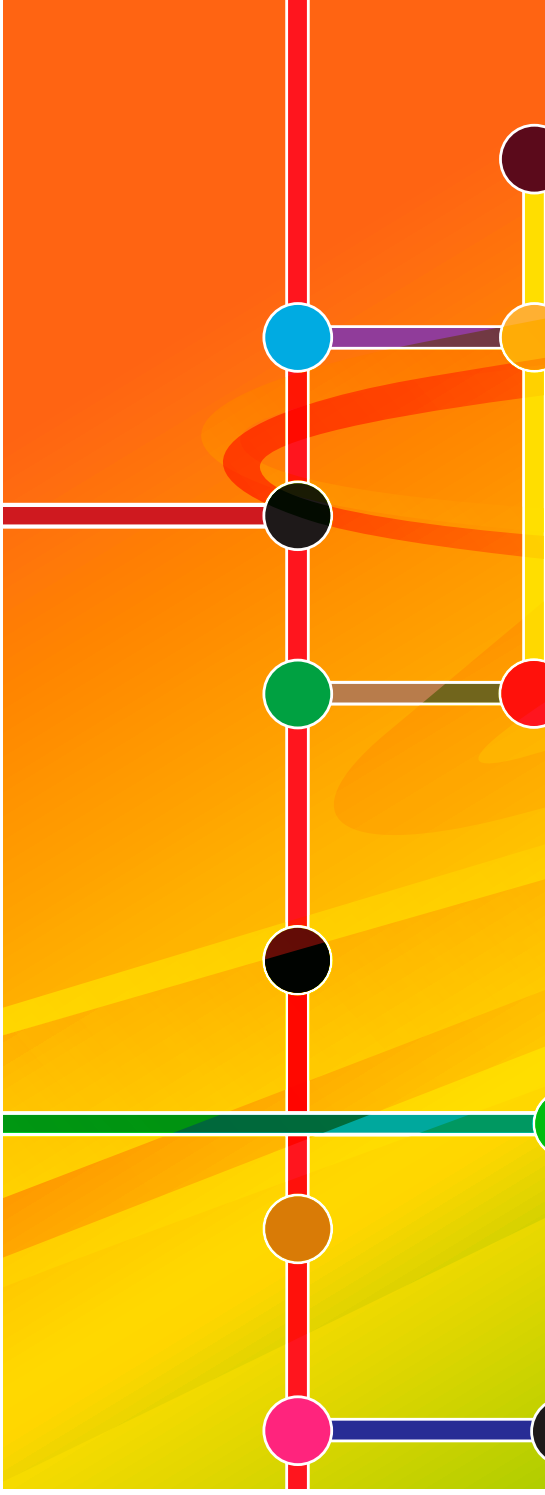
ShiViz is a new distributed system debugging visualization tool.

BY IVAN BESCHASTNIKH, PATTY WANG, YURIY BRUN, AND MICHAEL D. ERNST

Debugging Distributed Systems

DISTRIBUTED SYSTEMS POSE unique challenges for software developers. Reasoning about concurrent activities of system nodes and even understanding the system's communication topology can be difficult. A standard approach to gaining insight into system activity is to analyze system logs. Unfortunately, this can be a tedious and complex process. This article looks at several key features and debugging challenges that differentiate distributed systems from other kinds of software. The article presents several promising tools and ongoing research to help resolve these challenges.

Distributed systems differ from single-machine programs in ways that are simultaneously positive in providing systems with special capabilities, and negative in presenting software-development and operational challenges.



Heterogeneity. A distributed system's nodes may include mobile phones, laptops, server-class machines, and more. This hardware and software diversity in node resources and network connectivity can make a distributed system more robust, but this heterogeneity forces developers to manage compatibility during both development and debugging.

Concurrency. Simultaneous operation by multiple nodes leads to concurrency, which can make a distributed system outperform a centralized system. However, concur-

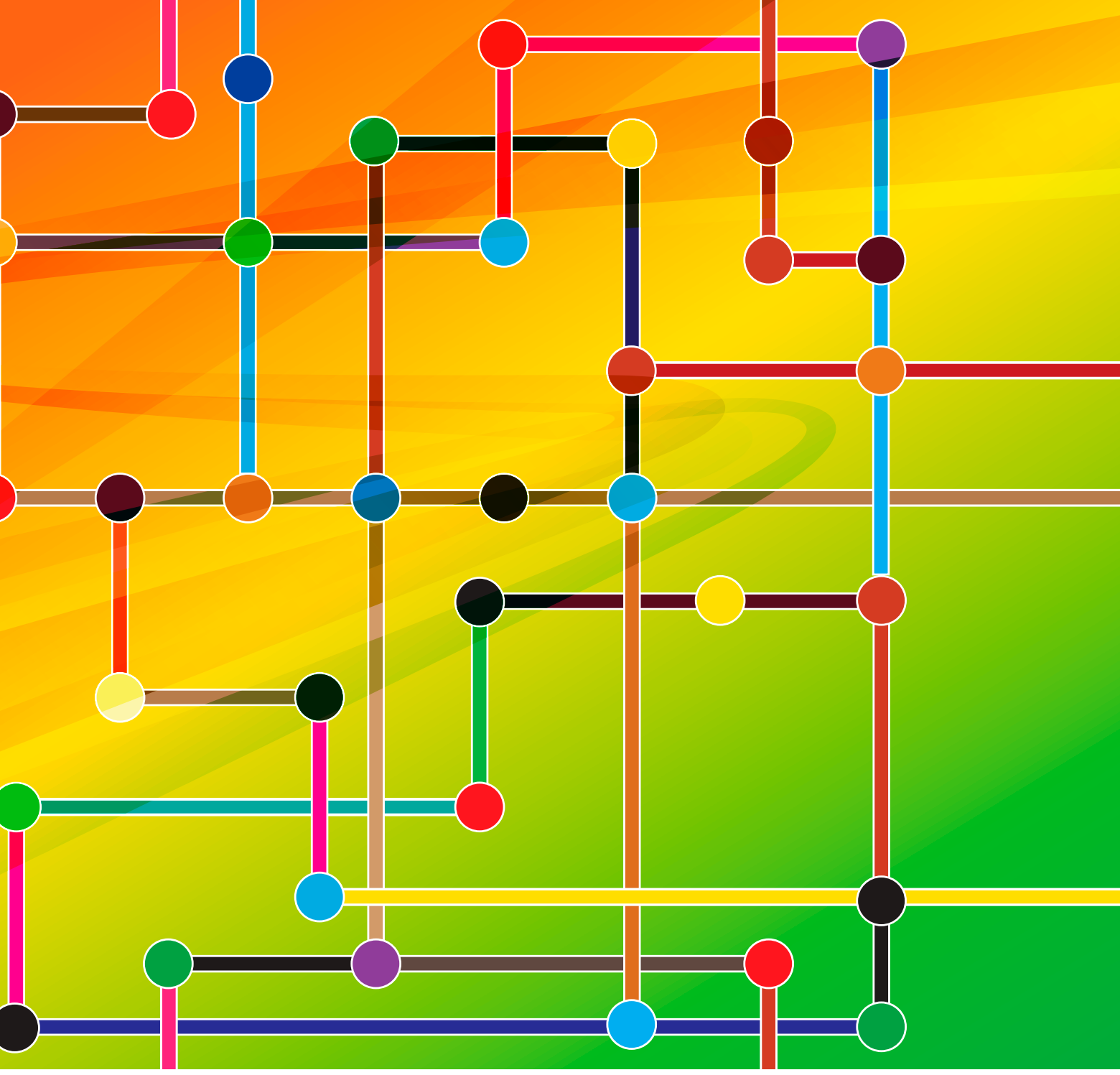


IMAGE FROM SHUTTERSTOCK.COM

rency may introduce race conditions and deadlocks, which are notoriously difficult to diagnose and debug. Additionally, networks introduce packet delay and loss, exacerbating the issues of understanding and debugging concurrency.

Distributing state. Distributing system state across multiple nodes can remove a central point of failure and improve scalability, but distributed state requires intricate node coordination to synchronize state across nodes—for example, nodes must ensure their local states are consistent. Potential inconsistencies are prevented by distributed algorithms, such as those that guarantee a particular flavor of data consistency and cache coherence. Developers may find it difficult, or even impossible, to reconstruct the global state of the system when it is distributed on many nodes. This complicates bug diagnosis and validation.

Partial failures. The distribution of state and responsibility allows distributed systems to be robust and survive a variety of failures. For example, Google's Spanner system can survive failures of entire data centers.[2] Achieving such fault tolerance, however, requires developers to reason through complex failure modes. For most distributed systems, fault tolerance cannot be an afterthought; the systems must be designed to deal with failures. Such failure resiliency is complex to design and difficult to test.

Existing Approaches

What follows is an overview of seven approaches designed to help software engineers validate and debug distributed systems.

Distributed Timestamps

A typical distributed-system log does not contain enough information to regenerate the happens-before relation, and this is one reason that distributed-system logs are so hard to interpret. ShiViz relies on logs that have been enhanced by another tool, ShiVector, to include vector clock timestamps that capture the happens-before relation between events.[10] Each node α maintains a vector of logical clocks, one clock for each node in the distributed system, including itself. α's *i*th clock is a lower bound on the current logical time at node *i*. The node α increments the αth component of its vector clock each time it performs a local action or sends or receives a message. Each message contains the sending node's current vector clock; upon message receipt, the receiving node updates its vector clock to the elementwise maximum of its local and received timestamps.

ShiVector is a lightweight instrumentation tool that augments the information already logged by a distributed system with the partial ordering information encoded as vector clocks. ShiVector interposes on communication and logging channels at each node in the system to add vector clock timestamps to every logged event.

ShiViz parses ShiVector-augmented logs to determine, for each event: the node that executed the event; the vector timestamp of the event; and the event's description. ShiViz permits a user to customize the parsing of logs using regular expressions, which can be used to associate additional information, or fields, with each event.

Testing. A test suite exercises a specific set of executions to ensure that they behave properly. Most testing of distributed systems is done using manually written tests, typically introduced in response to failures and then minimized.[14] Testing is an effective way to detect errors. However, since testing exercises a limited number of executions, it can never guarantee to reveal all errors.

Model checking is exhaustive testing, typically up to a certain bound (number of messages or steps in an execution). Symbolic model checking represents and explores possible executions mathematically; explicit-state model checking is more practical because it actually runs the program, controlling its executions rather than attempting to abstract it. MoDist performs black-box model checking, permuting message sequences and changing the execution speed of a process relative to other processes in the system.[18] MaceMC is a white-box technique that achieves speedups by adding programming-language support for model checking.[7] Common problems of all model-checking tools are scalability and environmental modeling, so they rarely achieve a guarantee.

Theorem proving can, in principle, prove a distributed system to be free of defects. Amazon uses TLA+ to verify its distributed systems.[11] Two recent systems can construct a verified distributed-system implementation. Verdi uses the Coq tool, whose expressive type system makes type checking equivalent to theorem proving, thanks to the Curry-Howard isomorphism; the Coq specification is then compiled into an OCaml implementation of the distributed system.[16] In contrast, IronFleet uses TLA and Hoare-logic verification to similarly produce a verified implementation of a distributed system.[5] The enormous effort needed to use these tools makes them most appropriate for new implementations of small, critical cores. Other techniques are needed for existing distributed systems.

Record and replay captures a single execution of the system so that this execution can be later replayed or analyzed. This is especially useful when debugging nondeterministic behaviors. A record-and-replay tool such as Friday[4] or D3S[8] captures all nondeterministic events so that an execution can be reproduced exactly. Recording a complex execution, however, may be prohibitively expensive and may change the behavior of the underlying system.

Tracing tracks the flow of data through a system, even across applications and protocols such as a database, Web server, domain-name server, load balancer, or virtual private network protocol.[13] For example, pivot tracing dynamically instruments Java-based systems to collect user-defined metrics at different points in the system and collates the resulting data to provide an inter-component view of the metrics over multiple executions.[9] Dapper is a lower-level tracing system used at Google to trace infrastructure services.[15] Tracing is more efficient than record and replay because it focuses on a specific subset of the data, but it requires instrumenting applications and protocols to properly forward, without consuming, the tracing metadata.

Log analysis is an even lighter-weight approach that works with systems that cannot be modified. It is a common black-box approach in which a system's console logs, debug logs, and other log sources are used to understand the system. For example, Xu et al. applied machine learning to logs to detect anomalies in Google infrastructure services.[17] Detailed logs from realistic systems contain a great deal of valuable detail, but they tend to be so large that they are overwhelming to programmers, who as a result cannot directly benefit from them.

Visualization. The complexity of distributed systems has inspired work on visualization of such systems to make them more transparent to developers. For example, Theia displays a visual signature that summarizes various aspects of a Hadoop execution, such as the execution's resource utilization.[3] These signatures can be used to spot anomalies and to compare executions. Tools such as Theia provide high-level summaries of a system's behavior. They do not, however, help a developer understand the underlying communication pattern in the system, including the distributed ordering of messages.

Visualizing Distributed-System Executions

As noted earlier, the ability to visualize distributed-system executions can help developers understand and debug their distributed systems. ShiViz is such a visualization tool, displaying distributed-system executions as interactive time-space diagrams that explicitly capture distributed ordering of messages and events in the system. This diagram reproduces the events and interactions captured in the execution log, making the ordering information explicit through a concise visualization. A developer can expand, collapse, and hide parts of the diagram, as well as search for particular interaction patterns. ShiViz is freely available as a browser application; any developer can visualize a log, without installing software or sending the log over the network.

To provide a rich and accurate visu-

alization of a distributed system's execution, ShiViz displays the *happens-before* relation. Given event *e* at node *n*, the happens-before relation indicates all the events that logically precede *e*. Other events might have already occurred at other nodes according to wall-clock time, but node *n* cannot tell whether those other events happened before or after *e*, and they do not affect the behavior of *e*. This partial order can rule out which events do not cause others, identify concurrent events, and help developers mentally replay parts of the execution.

Figure 1 illustrates an execution of the two-phase commit protocol with one transaction manager and two replicas.[1] This time-space diagram is a visualization of the underlying happens-before partial order, showing an execution with three nodes. Lines with arrows denote the partial ordering of events, each of which has an associated vector timestamp in brackets. (See the accompanying sidebar on timestamps.)

Figure 2 shows a screenshot of ShiViz visualizing an execution of a distributed data-store system called Voldemort.[12] In the middle of the screen is the time-space diagram, with time flowing from top to bottom. The colored boxes at the top represent nodes, and the vertical lines below them are the node timelines. Circles on each node's timeline represent events executed by that node. Edges connect events, representing the recorded happens-before relation: an event that is higher in the graph happened before an event positioned lower in the graph that it is connected to via a downward path. ShiViz augments the time-space diagram with operations to help developers explore distributed-system executions and corresponding logs. Figure 2 details some of these operations.

Understanding Distributed-System Executions

ShiViz helps developers to understand the relative ordering of events and the likely chains of causality between events, which is important for debugging concurrent behavior; to query for certain events and interaction patterns between hosts; and to identify structur-

Figure 1. Time-space diagram of an execution with three nodes.

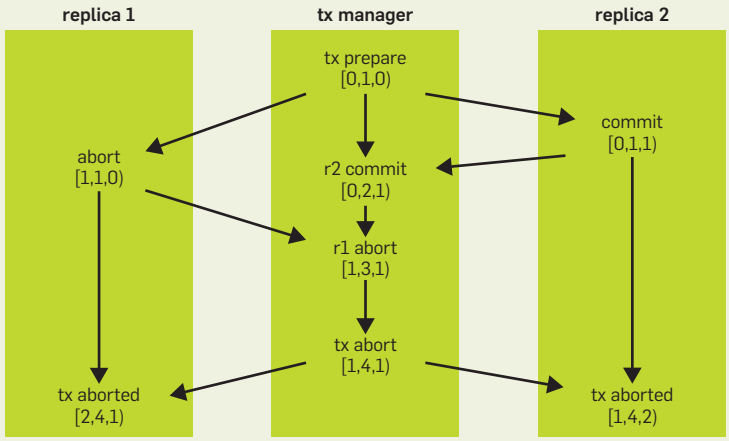


Figure 2. A ShiViz screenshot.

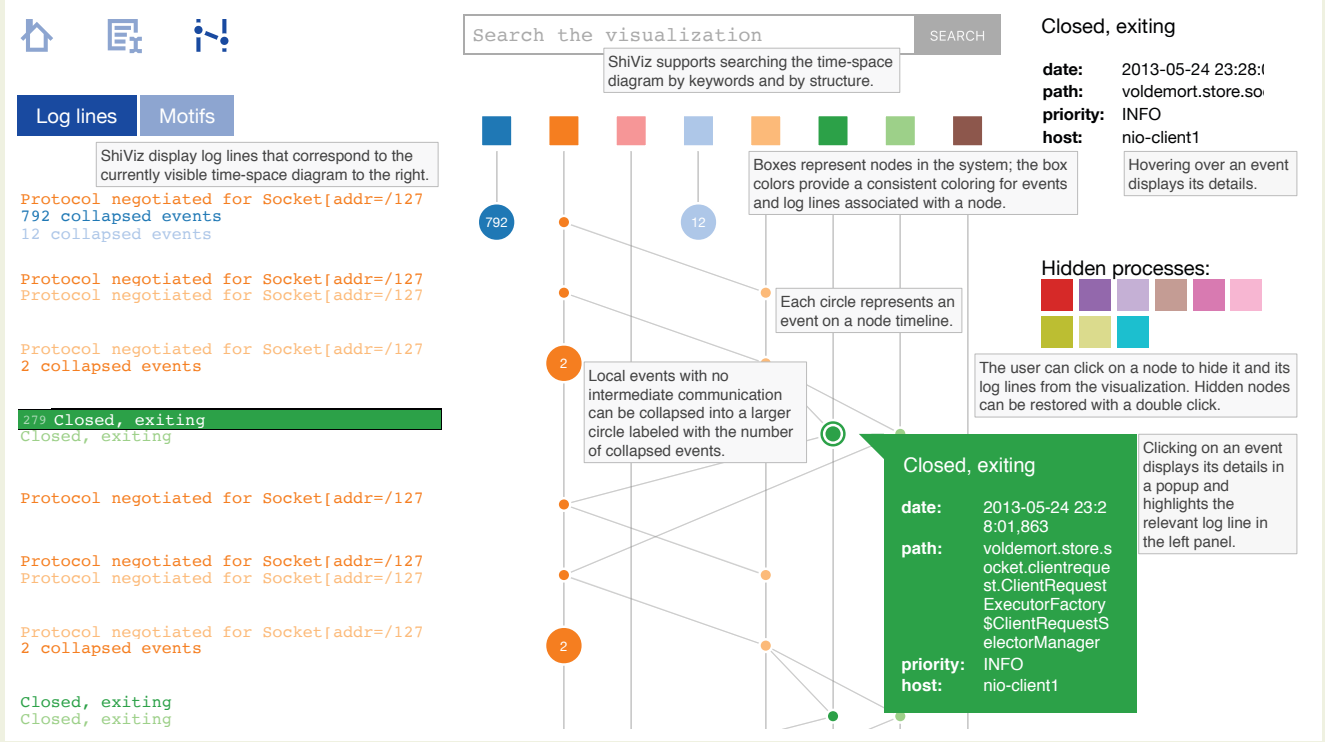


Figure 3. Structured search feature.

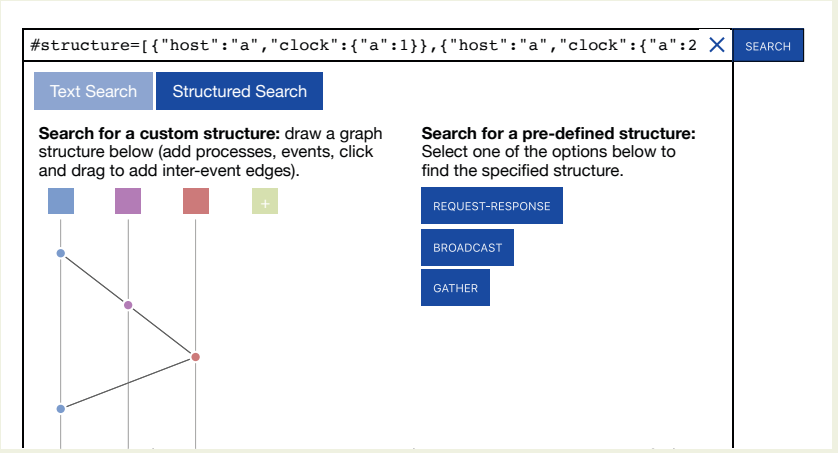
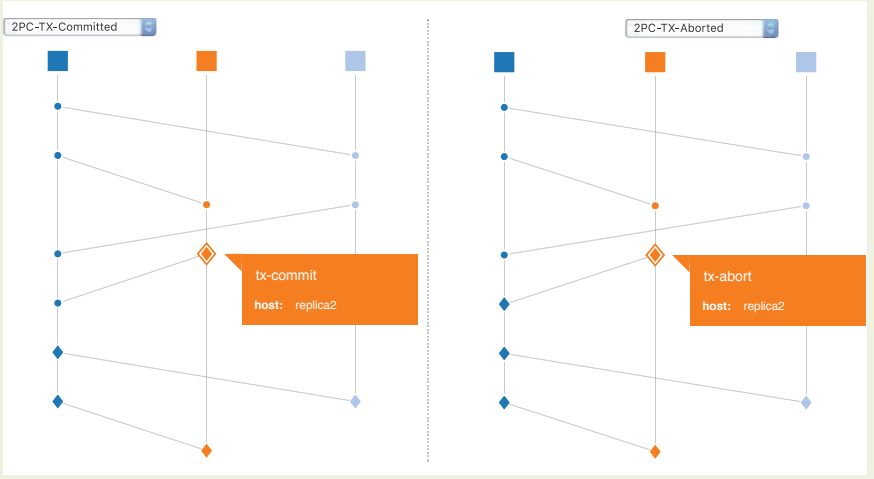


Figure 4. The two-phase commit protocol executions.



al similarities and differences between pairs and groups of executions. The time-space diagram representation supports the first goal by visualizing event ordering and communication. The next section describes two search operations that support the second goal, and operations over multiple executions that correspond to the third goal.

Keyword search and structured search operations. ShiViz implements two kinds of search operations: keyword and structured. Both types are accessible to the developer through the top search bar (see Figure 2).

Keyword search allows a developer to highlight all events in the diagram that contain a field matching a query. For example, searching for send will highlight all events in the diagram that have a field whose value is send. The results can be further constrained with field identifiers and regular expressions. For example, the query `node=alice && priority=CRITICAL*` will highlight only events at the `alice` node with a priority field matching the regular expression `CRITICAL*`.

In a structured search, a user queries ShiViz for any set of events related through a particular ordering pattern, and ShiViz highlights the sections of the diagram (events and their interconnections) that match this pattern. ShiViz includes several predefined patterns:

- *Request-response*. A source node sends a request and the destination node sends back a response.
- *Broadcast*. A node sends a message to most other nodes in the system.
- *Gather*. A node receives a message from most other nodes.

A user can also compose a custom pattern consisting of nodes, node events, and connections between events representing a partial order. Figure 3 shows such a custom pattern, depicting three nodes communicating in a ring: node 1 communicates only with node 2; node 2 with node 3; and node 3 with node 1. Drawing this pattern allows the user to search for all instances of this three-node ring communication in the execution. ShiViz automatically translates the drawn pattern into a textual representation (see search bar at the top), and it is possible to edit, copy, and paste the textual representation directly. The structured search feature allows users to express custom communication patterns between events and to query an execution for instances of the specified pattern. The presence or absence of queried subgraphs at particular points in an execution can help users detect anomalous behavior, aiding them in their debugging efforts.

Comparing executions. ShiViz can help users understand multiple executions of a system. When ShiViz parses multiple executions, the user can choose between viewing executions individually or pairwise.

In the pairwise view, a user can compare the two executions further by highlighting their differences. When enabled, the nodes are compared by name. For nodes present in both executions, ShiViz compares their events one by one by comparing the corresponding event descriptions. Nodes or events in one execution that do not appear in the other are redrawn as rhombuses.

Figure 4 illustrates this pairwise comparison on a log of the two-phase commit protocol. The two selected events in the figure explain the difference between these two executions: the two-phase commit successfully commits a transaction in the left execution, but aborts a transaction in the right execution.

The explicit highlighting of differences provides users with fast detection of anomalous events or points where the two executions diverge. The search features described earlier can be applied in the pairwise view to help developers detect specific unifying or distinguishing features across traces, allowing them to design and test their

systems more effectively.

Clustering executions. To help manage many executions, ShiViz supports grouping executions into clusters. A user can cluster by the number of nodes or by comparison to a base execution, using as a distance metric the differencing mechanism described earlier. Cluster results are presented as distinct groups of listed execution names.

Execution clusters aid in the inspection and comparison of multiple executions by providing an overview of all executions at once. Users can quickly scan through cluster results to see how executions are alike or different, based on the groups into which they are sorted. Clustering also helps users pinpoint executions of interest by allowing them to inspect a subset of executions matching a desired measure. This subset can be further narrowed by performing a keyword search or a structured search on top of the clustering results. Execution names among clusters are highlighted if their corresponding graphs contain instances matching the user's search query.

ShiViz helps developers visualize the event order, search for communication patterns, and identify potential event causality. This can help developers reason about the concurrency of events in an execution, distributed system state, and distributed failure modes, as well as formulate hypotheses about system behavior and verify them via execution visualizations. Meanwhile, the generality of logging makes ShiVector and ShiViz broadly applicable to systems deployed on a wide range of devices.

ShiViz has some limitations. ShiViz surfaces low-level ordering information, which makes it a poor choice for understanding high-level system behavior. The ShiViz visualization is based on logical and not realtime ordering, and cannot be used to study certain performance characteristics. The ShiViz tool is implemented as a client-side-only browser application, making it portable and appropriate for analyzing sensitive log data. This design choice, however, also limits its scalability. A related and complementary tool to ShiViz is Ravel, which can scalably visualize parallel execution traces.[6]

ShiViz is an open source tool with an online deployment (http://bestchai.bitbucket.org/shiviz/). Watch a video demonstrating key ShiViz features at http://bestchai.bitbucket.org/shiviz-demo/.

Acknowledgments

We thank Perry Liu and Albert Xing, who helped develop ShiViz; Jenny Abrahamson, who developed the initial ShiVector and ShiViz prototypes; and Donald Acton and Colin Scott, who helped evaluate ShiViz. This work is supported by NSERC USRA, the NSERC Discovery grant, and the National Science Foundation under grants CCF-1453474 and CNS-1513055. This material is based on research sponsored by DARPA under agreement number FA8750-12-2-0107. The U.S. government is authorized to reproduce and distribute reprints for governmental purposes, notwithstanding any copyright notices thereon. C

Related articles on queue.acm.org

Advances and Challenges in Log Analysis
Adam Oliner, Archana Ganapathi, and Wei Xu
http://queue.acm.org/detail.cfm?id=2082137

Leveraging Application Frameworks
Douglas C. Schmidt, Aniruddha Gokhale, and Balachandran Natarajan
http://queue.acm.org/detail.cfm?id=1017005

Postmortem Debugging in Dynamic Environments
David Pacheco
http://queue.acm.org/detail.cfm?id=2039361

References

1. Bernstein, P., Hadzilacos, V., Goodman, N. Distributed recovery. *Concurrency Control and Recovery in Database Systems*, Chapter 7. Addison-Wesley, 1986; http://research.microsoft.com/en-us/people/philbe/chapter7.pdf.
2. Corbett, J. C. et al. Spanner: Google's globally distributed database. In *Proceedings of the 10th Usenix Symposium on Operating Systems Design and Implementation*, 2012; https://www.usenix.org/conference/osdi12/technical-sessions/presentation/corbett.
3. Garduno, E., Kavulya, S. P., Tan, J., Gandhi, R., Narasimhan, P. Theia: Visual signatures for problem diagnosis in large Hadoop clusters. In *Proceedings of the 26th International Conference on Large Installation System Administration*, 2012, 33–42; https://users.ece.cmu.edu/~spertet/papers/hadoopvis-lisa12-cameraready-v3.pdf.
4. Geels, D., Altekar, G., Maniatis, P., Roscoe, T., Stoica, I. Friday: Global comprehension for distributed replay. In *Proceedings of the 4th Usenix Conference on Networked Systems Design and Implementation*, (2007); https://www.usenix.org/legacy/event/nsdi07/tech/full_papers/geels/geels.pdf.
5. Hawblitzel, C., Howell, J., Kapritsos, M., Lorch, J. R., Parno, B., Roberts, M. L., Setty, S., Zill, B. IronFleet: Proving practical distributed systems correct. In *Proceedings of the 25th Symposium on Operating Systems Principles*; 2015; http://sigops.org/sosp/sosp15/current/2015-Monterey/250-hawblitzel-online.pdf.
6. Isaacs, K.E. et al. Combing the communication hairball: Visualizing parallel execution traces using logical time. *IEEE Transactions on Visualization and Computer Graphics 20*, 12 (Dec 2014), 2349–2358.
7. Killian, C., Anderson, J. W., Jhala, R., Vahdat, A. Life, death, and the critical transition: Finding liveness bugs in systems code. In *Proceedings of the 4th Usenix Conference on Networked Systems Design and Implementation*, (2007); https://www.usenix.org/legacy/event/nsdi07/tech/killian/killian.pdf.
8. Liu, X., Guo, Z., Wang, X., Chen, F., Lian, X., Tang, J., Wu, M., Kaashoek, M. F., Zhang, Z. D3S: Debugging deployed distributed systems. In *Proceedings of the 5th Usenix Symposium on Networked Systems Design and Implementation*, 2008; 423–437; http://static.usenix.org/event/nsdi08/tech/full_papers/liu_xuezheng/liu_xuezheng.pdf.
9. Mace, J., Roelke, R., Fonseca, R. Pivot tracing: Dynamic causal monitoring for distributed systems. In *Proceedings of the 25th Symposium on Operating Systems Principles*, (2015); 378–393; http://sigops.org/sosp/sosp15/current/2015-Monterey/122-mace-online.pdf.
10. Mattern, F. Virtual time and global states of distributed systems. In *Proceedings of the International Workshop on Parallel and Distributed Algorithms*, 1989; http://homes.cs.washington.edu/~arvind/cs425/doc/mattern89virtual.pdf
11. Newcombe, C., Rath, T., Zhang, F., Munteanu, B., Brooker, M., Deardeuff, M. How Amazon Web Services uses formal methods. *Commun. ACM 58*, 4 (2015), 66–73; http://cacm.acm.org/magazines/2015/4/184701-how-amazon-web-services-uses-formal-methods/fulltext.
12. Project Voldemort; http://www.project-voldemort.com/voldemort/.
13. Sambasivan, R.R., Fonseca, R., Shafer, I., Ganger, G. So, you want to trace your distributed system? Key design insights from years of practical experience. Parallel Data Laboratory, Carnegie Mellon University, 2014; http://www.pdl.cmu.edu/PDL-FTP/SelfStar/CMU-PDL-14-102.pdf.
14. Scott, C. et al. Minimize faulty executions of distributed systems. In *Proceedings of the 13th Usenix Symposium on Networked Design and Implementation* (Santa Clara, CA, Mar. 16–18, 2016) 291–309.
15. Sigelman, B. H., Barroso, L. A., Burrows, M., Stephenson, P., Plakal, M., Beaver, D., Jaspan, S., Shanbhag, C. Dapper, a large-scale distributed systems tracing infrastructure. Research at Google, 2010; http://research.google.com/pubs/pub36356.html.
16. Wilcox, J. R., Woos, D., Panchekha, P., Tatlock, Z., Wang, X., Ernst, M. D., Anderson, T. Verdi: A framework for implementing and formally verifying distributed systems. In *Proceedings of the 36th SIGPLAN Conference on Programming Language Design and Implementation*, 2015, 357–368; https://homes.cs.washington.edu/~ztatlock/pubs/verdi-wilcox-pldi15.pdf.
17. Xu, W., Huang, L., Fox, A., Patterson, D., Jordan, M. Experience mining Google's production console logs. In *Proceedings of the Workshop on Managing Systems via Log Analysis and Machine Learning Techniques*, 2010; http://iiis.tsinghua.edu.cn/~weixu/files/slaml10.pdf.
18. Yang, J., et al. MoDist: Transparent model checking of unmodified distributed systems. In *Proceedings of the 6th Usenix Symposium on Networked Systems Design and Implementation*, 2009, 213–228; https://www.usenix.org/legacy/event/nsdi09/tech/full_papers/yang/yang_html/.

Ivan Beschastnikh (http://www.cs.ubc.ca/~bestchai/) works on improving the design, implementation, and operation of complex systems. He is an assistant professor in the department of computer science at the University of British Columbia, where he leads a team of students on projects that span distributed systems, software engineering, security, and networks, with a particular focus on program analysis.

Patty Wang has explored approaches to helping developers understand and compare multiple distributed executions, focusing on summarizing similarities and differences across traces.

Yuriy Brun (http://people.cs.umass.edu/~brun/) works on automating system building and creating self-adaptive systems. He is an assistant professor at the University of Massachusetts, Amherst.

Michael D. Ernst (http://homes.cs.washington.edu/~mernst/) researches ways to make software more reliable, more secure, and easier to produce. His primary technical interests are in software engineering, programming languages, type theory, and security, among others.

DOI:10.1145/2948983

Article development led by acmqueue
queue.acm.org

SQL has a brilliant future as a major figure in the pantheon of data representations.

BY PAT HELLAND

The Singular Success of SQL

SQL HAS BEEN singularly successful in its impact on the database industry. Nothing has come remotely close to its ubiquity. Its success comes from its high-level use of relational algebra allowing set-oriented operations on data shaped as rows, columns, cells, and tables.

SQL's impact can be seen in two broad areas. First, the programmer can accomplish a lot very easily with set-oriented operations. Second, the high-level expression of the programmer's intent has empowered huge performance gains.

This article discusses how these features are dependent on SQL creating a notion of stillness through transactions and a notion of a tight group of tables with schema fixed at the moment of the transaction. These characteristics are what make SQL different from the increasingly pervasive distributed systems.

SQL has a brilliant past and a brilliant future. That future is not as the singular and ubiquitous holder of data but rather as a major figure in the pantheon of data representations. What the heck happens when data is not kept in SQL?

SQL: The Miracle of the Age, of the Ages, and of the Aged

I launched my career in database implementation when Jimmy Carter was president. At the time, there were a couple of well-accepted representations for data storage: the *network* model was expressed in the CODASYL (Conference/Committee on Data Systems Languages) standard with data organized in sets having one set owner (parent) and multiple members (children); the *hierarchical* model ensured all data was captured in a tree structure with records having a parent-child-grandchild relationship. Both of these models required the programmer to navigate from record to record.

Then along came these new-fangled relational things. INGRES (and its language QUEL) came from UC Berkeley. System-R (and its language SQL) came from IBM Research. Both leveraged relational algebra to support set-oriented abstractions allowing powerful access to data.

At first, they were really, really, really slow. I remember lively debates with database administrators who fervently believed they must be able to know the cylinder on disk holding their records! They most certainly did not want to change from their hierarchical and

IMAGE BY FOCAL POINT


SQL

network databases. As time went on, SQL became inexorably faster and more powerful. Soon, SQL meant database and database meant SQL.

A funny thing happened by the early 2000s, though. People started putting data in places other than "the database." The old-time database people (including yours truly) predicted their demise. Boy, were we wrong!

Of course, for most of us who had worked so hard to build transactional, relational, and strongly consistent systems, these new representations of data in HTML, XML, JSON, and other formats didn't fit into our worldview. The radicals of the 1970s and 1980s became the fuddy-duddies of the 2000s. A new schism had emerged.

Distributed transactions across different SQL databases are rare and challenging.

SQL, Values, and Relational Algebra

Relational databases have tables with rows and columns. Each column in a row provides a cell that is of a well-known type. Data Definition Language (DDL) specifies the tables, rows, and columns and can be dynamically changed at any time, transforming the shape of the data.

The fundamental principle in the relational model is that all interrelating is achieved by means of comparisons of values, whether these values identify objects in the real world or indicate properties of those objects. A pair of values may be meaningfully compared, however, if and only if these values are drawn from a common domain.

The stuff being compared in a query must have matching DDL or it doesn't make sense. SQL depends on its DDL being rigid for the duration of the query.

There is not really a notion of some portion of the SQL data having extensible metadata that arrives with the data. All of the metadata is defined before the query is issued. Extensible data is, by definition, not defined (at least at the receiver's system).

SQL's strength depends on a well-defined schema. Its set-oriented nature uses the well-defined schema for the duration of the operations. The data and metadata (schema) must remain still while SQL does its thing.

The Stillness and Isolation of Transactions

SQL is set oriented. Bad stuff happens when the set of data slides around during the computation. SQL is supposed to produce consistent results. Those consistent results are dependent on input data that *appears to be unchanging*.

Transactions and, specifically, transactional isolation provide the sense that nothing else is happening in the world.

The Holy Grail of transaction isolation is *serializability*. The idea is to make transactions *appear* as if they happened in a serial order. They don't actually have to occur in a serial order; it just has to seem like they do.

In the accompanying figure, the red transaction Ti depends upon changes made by the green transactions (Ta, Tb, Tc, Td, and Tf). The blue transactions (Tk, Tl, Tm, Tn, and To) depend on the changes made by Ti. Ti definitely is ordered after the green transactions and before the blue ones. It doesn't matter if any of the yellow transactions (Te, Tg, Tj, and Th) occur before or after Ti. There are many correct serial orders. What matters is the concurrency implemented in the system provides a view that is serializable.

Suddenly, the world is still and set orientation can smile on it.

A Sense of Place

SQL and its relational data are almost always kept inside a single system or a few systems close to each other. Each SQL database is almost always contained within a trust boundary and protected by surrounding application code.

I don't know of any systems that allow untrusted third parties to access their back-end databases. My bank's ATM, for example, has never let me directly access its back-end database with Java Database Connectivity (JDBC). So far, the bank has constrained me to a handful of operations such as deposit, withdrawal, or transfer. It's really annoying! In fact, I can't think of any enterprise databases that allow untrusted third parties to "party" on their databases. All of them insist on using application code to mitigate the foreigners' access to the system.

Interactions occur across these systems, but they are implemented with some messages or other data exchange that is loosely coupled to the underlying databases on each side. The messages hit the application code and not the database.

Each of these databases appears to be an island unto itself. Now, that island may have a ferry or even a four-lane bridge connecting it to other islands. Still, you always know the island upon which you stand when you access a database.

Different Places Means Different Times

Multiple databases sharing a transactional scope is extremely rare. When a transaction executes on a database, there is no clear and crisp notion of its time when compared with the time on another system's database. Distributed transactions across different SQL databases are rare and challenging.

If you assume two databases do not share a transactional scope, then the simple act of spreading work across space (the databases) implies spreading the work across time (multiple transactions). This transition from one database to more than one database is a seminal semantic shift. Space and time are intrinsically tied to each other.

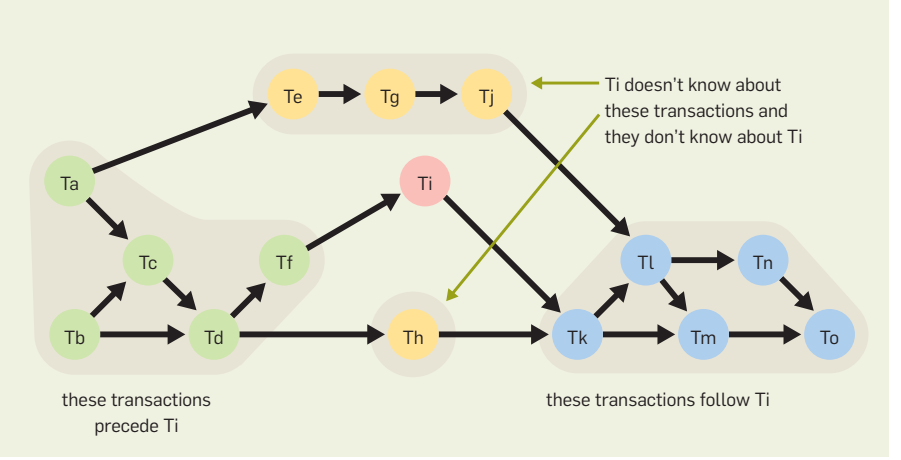
When you pop from one to many, SQL and its relational algebra cannot function without some form of restriction. The most common form is to cast some of the data into immutable views that can be meaningfully queried over time. The system projecting these views won't change them. When they don't change, you can use them across space.

Freezing data in time allows its use across spatial boundaries. Typically, you also project the data to strip out the private stuff as you project across trust boundaries. To span spatial boundaries, time must freeze, at least for the data being shared. When you freeze data, it's immutable.

Immutability: The One Constant of Distributed Systems Immutable data can be immortal and omnipresent. Think of it as the Gideon Bible, which seems to be in every hotel room; I suspect there will be Bibles there for a long time. If you want to do a query leveraging the Gideon Bible as an input, you will not struggle with challenges of concurrency or isolation. It's relatively straightforward to cache a copy close to where you need it, too.

SQL's relational operations can be applied to immutable data at a massive scale because the metadata is immutable and the data is immutable. This empowers MapReduce, Hadoop, and the other big-data computation. By being immutable, the contents are still and the set-oriented computations make sense.

Transaction serializability.



Immutable data can be everywhere at any time. That allows it to be both inside the singularity and outside of it. No big deal. Immutability truly is one of the unifying forces of distributed systems.

Classic centralized databases force their data to appear immutable using transactions. When distribution impedes the use of transactions, you snapshot a subset of your data so it can be cast across the boundaries with predictable behavior.

Escaping the Singularity

SQL databases are phenomenally powerful and have enjoyed singular success in providing access to and control over data. They allow the combination and analysis of data by leveraging relational algebra. Relational algebra relates values contained in the rows and the columns of its tables. This has provided incredible power in programming and huge performance gains in accessing relational data.

To do this, relational algebra requires a static set of tables unmolested by concurrent changes. Both the data and the schema for the data must be static while operations are performed. This is achieved with transactional serializability or other slightly weaker isolation policies. Serializability provides the illusion that each user of the database is alone at a *single point in time*.

In a relational database, it is difficult to provide full functionality when distributed except, perhaps, across a handful of machines in close proximity. Even more profoundly, SQL works well within a single trust boundary such as a department or a company. SQL databases provide the illusion that they exist at a *single point in space*.

Providing a single point in space and time yields both stillness and isolated location. This empowers the value-based comparisons of relational algebra. It looks just like a *singularity*.

The industry has leapt headlong toward data representations that are neither bound to a single point in time nor to a single point in space with distributed, heterogeneous, and loosely coupled systems. Nowadays, far more data is being generated outside the SQL environment than within it. This trend is accelerating.

Future articles will explore various consequences of *escaping the singularity* and relaxing the constraints of both space and time.

No, it ain't your grandmother's database anymore. C

Related articles on queue.acm.org

Scalable SQL
Michael Rys
http://queue.acm.org/detail.cfm?id=1971597

If You Have Too Much Data, then "Good Enough" Is Good Enough
Pat Helland
http://queue.acm.org/detail.cfm?id=1988603

All Your Database Are Belong to Us
Erik Meijer
http://queue.acm.org/detail.cfm?id=2338507

Pat Helland has been implementing transaction systems, databases, application platforms, distributed systems, fault-tolerant systems, and messaging systems since 1978. He currently works at Salesforce.

DOI:10.1145/2948985

Article development led by acmqueue
queue.acm.org

Microservices aren't for every company, and the journey isn't easy.

BY TOM KILLALEA

The Hidden Dividends of Microservices

MICROSERVICES ARE AN approach to building distributed systems in which services are exposed only through hardened APIs; the services themselves have a high degree of internal cohesion around a specific and well-bounded context or area of responsibility, and the coupling between them is loose. Such services are typically simple, yet they can be composed into very rich and elaborate applications. The effort required to adopt a microservices-based approach is considerable, particularly in cases that involve migration from more monolithic architectures. The explicit benefits of microservices are well known and numerous, however, and can include increased agility, resilience, scalability, and developer productivity. This article identifies some of the hidden dividends of microservices that implementers should make a conscious effort to reap.

The most fundamental of the benefits driving the momentum behind microservices is the clear separation of concerns, focusing the attention of each service upon some well-defined aspect of the overall application. These services can be composed in novel ways with loose coupling between the services, and they can be deployed independently. Many implementers are drawn by the allure of being able to make changes more frequently and with less risk of negative impact. Robert C. Martin described the *single responsibility principle*: "Gather together those things that change for the same reason. Separate those things that change for different reasons."[5] The clear separation of concerns, minimal coupling across domains of concern, and the potential for a higher rate of change lead to increased business agility and engineering velocity.

Martin Fowler argues the adoption of continuous delivery and the treatment of infrastructure as code are more important than moving to microservices, and some implementers adopt these practices on the way to implementing microservices, with positive effects on resilience, agility, and productivity. An additional key benefit of microservices is they can enable owners of different parts of an overall architecture to make very different decisions with respect to the hard problems of building large-scale distributed systems in the areas of persistence mechanism choices, consistency, and concurrency. This gives service owners greater autonomy, can lead to faster adoption of new technologies, and can allow them to pursue custom approaches that might be optimal for only a few or even for just one service.

The Dividends

While difficult to implement, a microservices-based approach can pay dividends to the organization that takes the trouble, though some of the benefits are not always obvious. What follows is a description of a few of the less obvious ones that may make the adoption of microservices worth the effort.

IMAGE BY SASHKIN

Dividend #1: Permissionless Innovation

Permissionless innovation is about "the ability of others to create new things on top of the communications constructs that we create,"[1] as put forth by Jari Arkko, chair of the Internet Engineering Task Force (IETF). When enabled, it can lead to innovations by consumers of a set of interfaces that the designers of those interfaces might find surprising and even bewildering. It contrasts with approaches where *gatekeepers* (a euphemism for *blockers*) have to be consulted before an integration can be considered.

To determine whether permissionless innovation has been unleashed to the degree possible, a simple test is to look at the prevalence of meetings *between* teams (as distinct from *within* teams). Cross-team meetings suggest coordination, coupling, and problems with the granularity or functionality of service interfaces. Engineers do not seek out meetings if they can avoid them; such meetings could mean that a service's APIs are not all that is needed to integrate. An organization that has embraced permissionless innovation should have a high rate of experimentation and a low rate of cross-team meetings.

Dividend #2: Enable Failure

It should come as no surprise to hear that in computer science, we still don't know how to build complex systems that work reliably,[6] and the unreliability of systems increases with size and complexity. While opinions differ as to whether microservices allow a reduction in overall complexity, it's worth embracing the notion that microservices will typically increase the number of failures. Further, failures across service boundaries will be more difficult to troubleshoot since external call stacks are inherently more fragile than internal ones, and the debugging task is limited by poorer tooling and by more challenging ad hoc analysis characteristics. This tweet by @Honest_Update can sometimes feel uncomfortably accurate: "We replaced our monolith with micro services so that every outage could be more like a murder mystery."[4]

Designing for the inevitability and indeed the routineness of failure can lead to healthy conversations about state persistence, resilience, dependency management, shared fate, and graceful degradation. Such conversations should lead to a reduction of the blast radius of any given failure by leveraging techniques such as caching, metering, traffic engineering, throttling, load shedding, and backoff. In a mature microservices-based architecture, failure of individual services should be expected, whereas the cascading failure of all services should be impossible.

Dividend #3: Disrupt Trust

In small companies or in small code bases, some engineers may have a strong sense of trust in what is being deployed because they look over every shoulder and review every commit. As team size and aggregate velocity increase, "Dunbar's number" takes effect, leading to such trust becoming strained. As defined by British anthropologist Robin Dunbar, this is the maximum number of individuals with whom one can maintain social relationships by personal contact.

A move to microservices can force this expectation of trust to surface and be confronted. The boundary between one service and another becomes a set of APIs. The consumer gives up influence over the design of what lies behind those APIs, how that design evolves, and how its data persists, in return for a set of SLAs (service-level agreements) governing the stability of the APIs and their runtime characteristics. Trust can be replaced with a combination of autonomy and accountability.

Microservices encourage the "you build it, you own it" model.

As stated by Melvin Conway, who defined what is now known as Conway's law: "Any organization that designs a system will inevitably produce a design whose structure is a copy of the organization's communication structure."[2]

Microservices can provide an effective model for evolving organizations that scale far beyond the limits of personal contact.

Dividend #4: You Build It, You Own It

Microservices encourage the "you build it, you own it" model. Amazon CTO Werner Vogels described this model in a 2006 conversation with Jim Gray that appeared in *ACM Queue*: "Each service has a team associated with it, and that team is completely responsible for the service—from scoping out the functionality, to architecting it, to building it, and operating it. You build it, you run it. This brings developers into contact with the day-to-day operation of their software. It also brings them into day-to-day contact with the customer. The customer feedback loop is essential for improving the quality of the service."[3]

In the decade since that conversation, as more software engineers have followed this model and taken on responsibility for the operation as well as the development of microservices, they have driven broad adoption of a number of practices that enable greater automation and that lower operational overhead. Among these are continuous deployment, virtualized or containerized capacity, automated elasticity, and a variety of self-healing techniques.

Dividend #5: Accelerate Deprecations

In a monolith, it's difficult to deprecate anything safely. With microser-

vices, it's easy to get a clear view of a service's call volume, to stand up different and potentially competing versions of a service, or to build a new service that shares nothing with the old service other than backward compatibility with those interfaces that consumers care about the most.

In a world of permissionless innovation, services can and should routinely come and go. It's worth investing some effort to make it easier to deprecate services that have not meaningfully caught on. One approach to doing this is to have a sufficiently high degree of competition for resources so that any resource-constrained team that is responsible for a languishing service is drawn to spending most of their time on other services that matter more to customers. As this occurs, responsibility for the unsuccessful service should be transferred to the consumer who cares about it the most. This team may rightfully consider themselves to have been left "holding the can," although the deprecation decision also passes into their hands. Other teams that wish not to be left holding the can have an added incentive to migrate or terminate their dependencies. This may sound brutal, but it's an important part of "failing fast."

Dividend #6: End Centralized Metadata

In Amazon's early years, a small number of relational databases were used for all of the company's critical transactional data. In the interest of data integrity and performance, any proposed schema change had to be reviewed and approved by the DB Cabal, a gatekeeping group of well-meaning enterprise modelers, database administrators, and software engineers. With microservices, consumers should not know or care about how data persists behind a set of APIs on which they depend, and indeed it should be possible to swap out one persistence mechanism for another without consumers noticing or needing to be notified.

Dividend #7: Concentrate The Pain

A move to microservices should enable an organization to take on very different approaches to the governance expectations that it has of different services. This will start with a consistent companywide model for data classification and with the classification of the criticality of the integrity of different business processes. This will typically lead to threat modeling for the services that handle the most important data and processes, and the implementation of the controls necessary to serve the company's security and compliance needs. As microservices proliferate, it can be possible to ensure the most severe burden of compliance is concentrated in a very small number of services, releasing the remaining services to have a higher rate of innovation, comparatively unburdened by such concerns.

Dividend #8: Test Differently

Engineering teams often view the move to microservices as an opportunity to think differently about testing. Frequently, they will start thinking about how to test earlier in the design phase, before they start to build their service. A clearer definition of ownership and scope can provide an incentive to achieve greater coverage. As stated by Yelp in setting forth its service principles, "Your interface is the most vital component to test. Your interface tests will tell you what your client actually sees, while your remaining tests will inform you on how to ensure your clients see those results."[7]

The adoption of practices such as continuous deployment, smoke tests, and phased deployment can lead to tests with higher fidelity and lower time-to-repair when a problem is discovered in production. The effectiveness of a set of tests can be measured less by their rate of problem detection and more by the rate of change that they enable.

Warning Signs

The following indicators are helpful in determining that the journey to microservices is incomplete. You are probably not doing microservices if:

- Different services do coordinated deployments.
- You ship client libraries.
- A change in one service has unexpected consequences or requires a change in other services.
- Services share a persistence store.
- You cannot change your service's persistence tier without anyone caring.
- Engineers need intimate knowledge of the designs and schemas of other teams' services.
- You have compliance controls that apply uniformly to all services.
- Your infrastructure isn't programmable.
- You can't do one-click deployments and rollbacks.

Conclusion

Microservices aren't for every company, and the journey isn't easy. At times the discussion about their adoption has been effusive, focusing on autonomy, agility, resilience, and developer productivity. The benefits don't end there, however, and to make the journey worthwhile, it's important to reap the additional dividends. C

Related articles on queue.acm.org

A Conversation with Werner Vogels
http://queue.acm.org/detail.cfm?id=1142065

The Verification of a Distributed System
Caitie McCaffrey
http://queue.acm.org/detail.cfm?id=2889274

There's Just No Getting around It: You're Building a Distributed System
Mark Cavage
http://queue.acm.org/detail.cfm?id=2482856

References

1. Arkko, J. Permissionless innovation. IETF; https://www.ietf.org/blog/2013/05/permissionless-innovation/.
2. Conway, M.E. How do committees invent? *Datamation Magazine* (1968); http://www.melconway.com/Home/Committees_Paper.html.
3. Gray. J. A conversation with Werner Vogels. *ACM Queue 4*, 4 (2006); http://queue.acm.org/detail.cfm?id=1142065.
4. Honest Status Page. @honest_update, 2015; https://twitter.com/honest_update/status/651897353889259520.
5. Martin, R.C. The single responsibility principle; http://blog.8thlight.com/uncle-bob/2014/05/08/SingleReponsibilityPrinciple.html.
6. Perera, D. The crypto warrior. Politico; http://www.politico.com/agenda/story/2015/12/crypto-war-cyber-security-encryption-000334.
7. Yelp service principles; https://github.com/Yelp/service-principles.

Tom Killalea was with Amazon for 16 years and now consults and sits on several company boards, including those of Capital One, ORRECO, and MongoDB.

DOI:10.1145/2858789

The aim is to improve cities' management of natural and municipal resources and in turn the quality of life of their citizens.

BY RIDA KHATOUN AND SHERALI ZEADALLY

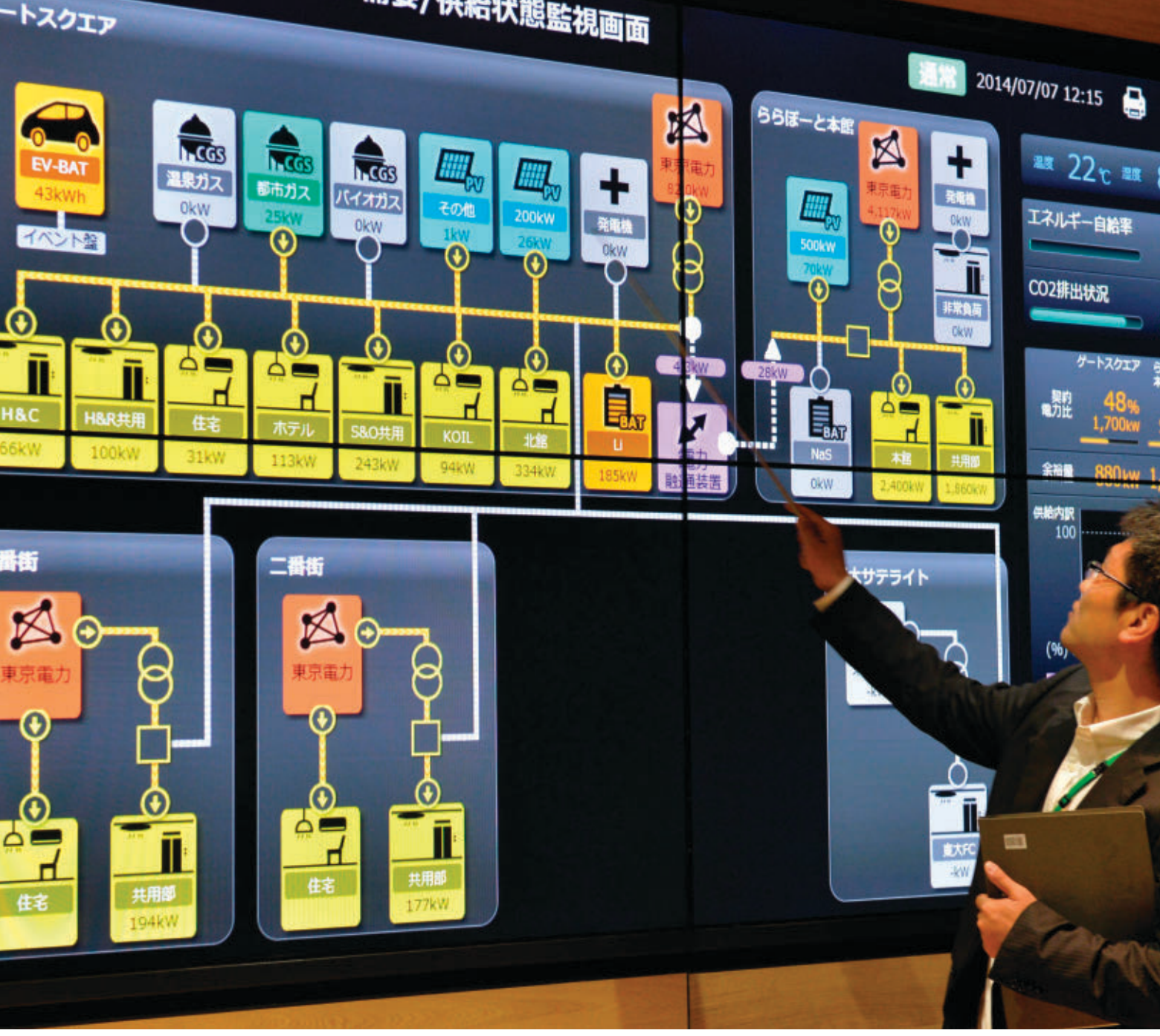
Smart Cities: Concepts, Architectures, Research Opportunities

BY 2030, THE world's population is projected to be 8.5 billion and increase to 9.7 billion by 2050 and 11.2 billion by 2100. Half of humanity today lives in cities. Many cities are experiencing exponential growth as people move from rural areas in search of better jobs and education. Consequently, cities' services and infrastructures are being stretched to their limits in terms of scalability, environment, and security as they adapt to support this population growth. Visionaries and planners are thus seeking a sustainable, post-carbon economy[20] to improve energy efficiency and minimize carbon-emission levels. Along with cities' growth, innovative solutions are crucial for improving productivity (increasing operational efficiencies) and reducing management costs.

A smart city is an ultra-modern urban area that addresses the needs of businesses, institutions, and especially citizens. Here we should differentiate between a smart city and smart urbanism. The objective of these concepts is the same—the life of citizens. The architects of ancient cities did not take into consideration long-term scalability—housing accessibility, sustainable development, transport systems, and growth—and there is no scalable resource management that may be applied from one decade to another. Unfortunately, smart urbanism is not well represented in smart cities' development. Smart urbanism must also be considered as an aspect of a smart city, including information-communication technologies. In recent years, a significant increase in global energy consumption and the number of connected devices and other objects has led government and industrial institutions to deploy the smart city concept. Cities' demographic, economic, social, and environmental conditions are the major reasons for the dramatic increase in pollution, congestion, noise, crime, terrorist attacks, energy production, traffic accidents, and climate change. Cities today are the major contributors to the climate problem. They cover less than 2% of the Earth's surface yet consume 78% of the world's energy,

» key insights

- **Many cities in Asia, Europe, and North America are pursuing smart city projects.**
- **A smart city is a complex system, meaning even a single vulnerability could affect all citizens' security.**
- **Future research must address high energy consumption, security, privacy, lack of investment, smart citizens, and other related challenges to enable secure, robust, scalable smart city development and adoption.**



An operator checks the electric power control panel for the smart city project in Kashiwa, Japan, on July 7, 2014.

PHOTO BY YOSHIKAZU TSUNO/AFP/GETTY IMAGES

producing more than 60% of all CO_2 emissions (http://unhabitat.org/).

Innovative solutions are imperative to address cities' social, economic, and environmental effects. Those solutions involve three key objectives:

Optimized management of energy resources. This objective could be realized through the Internet of Energy (IoE), or smart grid technology. The IoE[a,b] connects energy grids to the Internet, dispatching units of energy as needed, representing a set of distributed renewable electricity generators linked and managed through the Internet. The IoE enables accurate, real-time monitoring and optimization of power flows;

Decentralized energy production. The IoE concept allows consumers to be energy producers themselves, using renewable energy sources and combined heat and power units; decentralization enables smarter demand-response management of consumers' energy use; and

Integrated business models and economic models. These models describe how organizations should deliver and reap benefits from their services (such as transport, energy consumption, and charging tolls); such models must be designed to support city development.

Note the "smart city" label is not a marketing slogan. A city is "smart" if it provides better efficiency for urban planning through a variety of technologies. Smart cities are also defined, according to Anthony Townsend in his book *Smart Cities* (W.W. Norton & Company, 2014), as "places where information technology is combined with infrastructure, architecture, everyday objects, and our bodies to address social, economic, and environmental

a http://www.artemis-ioe.eu/ioe_consortium_area.htm
b http://www.bdi.eu/BDI_english/download_content/Marketing/Brochure_Internet_of_Energy.pdf

problems." The European Parliament proposed[c] this definition: "A smart city is a city seeking to address public issues via information and communication technology (ICT)-based solutions on the basis of a multi-stakeholder, municipality based partnership." This is quite broad, encompassing many fields, while the Japanese definition is more specific, focusing on energy, infrastructure, ICT, and lifestyle. From these definitions, we deduce ICT plays a pivotal role in developing a city that can adapt to the needs of its citizens. By leveraging advanced power systems, networking, and communication technologies, a smart city aims to enhance the lives of its citizens and optimize territorial, economic, and environmental resources. Smart cities promise multiple benefits:

Safety and security. This includes surveillance cameras, enhanced emergency-response services, and automated messages for alerting citizens; real-time information about a city should be available;

Environment and transportation. This entails controlled pollution levels, smart street lights, congestion rules, and new public-transport solutions to reduce car use;

Home energy management. Options include timely energy billing, optimal energy management, saving, perhaps, 30%–40% on electricity bills; the European Commission estimates approximately 72% of European electricity consumers will have smart meters by 2020;

Educational facilities. More investment is needed to improve educational opportunities for all, lifelong learning, education through remote learning, and smart devices in classrooms;

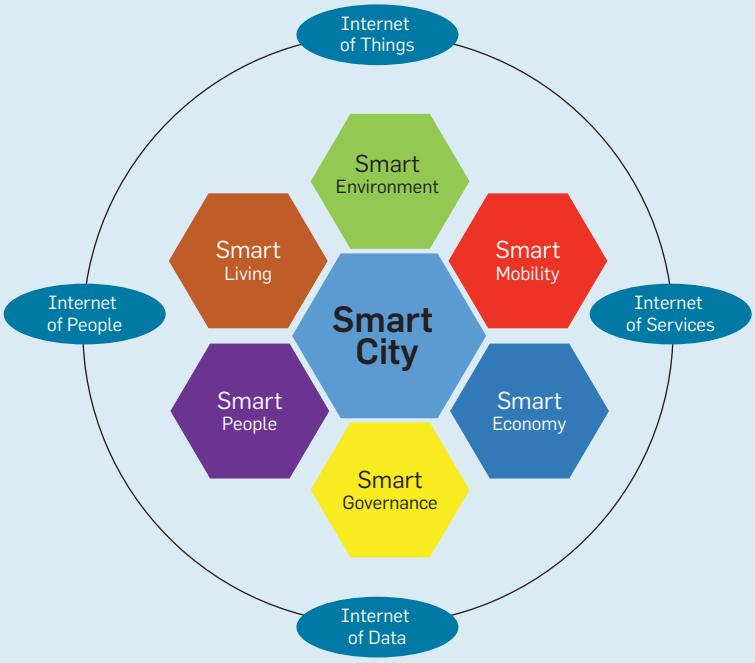
Tourism. Preserving a city's natural resources promotes the growth of tourism; additionally, smart devices offer direct and localized access to information;

Citizens' health. Using new technologies could improve people's health; citizens need full access to high-quality, affordable healthcare, and wireless body-area network technology—including sensors attached to the body or clothes and implanted under the skin—can acquire health information (such as heartbeat, blood sugar, and blood pressure) and transmit it in real time or offline through a smartphone to remote servers accessible by healthcare professionals for monitoring or treatment.

Despite this potential, many elements must be understood and considered before cities are able to reap the benefits. Here, we describe some of the basic concepts and architectural components of smart cities, including energy management (ISO 50001), smart homes, vehicular networks, smart grids, and quality of life (ISO 37120). We then examine recent smart city projects around the world, identifying some of the challenges and future research opportunities. We also highlight some of the risks introduced through information systems in the urban environment.

c http://www.smartcities.at/assets/Publikationen/Weitere-Publikationen-zum-Thema/mappingsmartcities.pdf

Figure 1. A smart city model.



Implementation and Deployment

Designing and deploying smart cities needs experts from multiple fields, including economics, sociology, engineering, ICT, and policy and regulation. Various frameworks describing the architecture of smart cities have been proposed by both industry and academic sources. One of the most widely adapted and adopted models is the reference model proposed by the U.S. National Institute of Standards and Technology. Smart cities are complex systems, often called "systems of systems," including people, infrastructure, and process components (see Figure 1). Most smart cities models consist of six components: government, economy, mobility, environment, living, and people. The European Parliament Policy Department said in 2014 that 34% of smart cities in Europe have only one such component.

Multiple approaches and methods have been proposed to evaluate smart cities from multiple perspectives, including an urban Internet of Things (IoT) system for smart cities, sustainability, global city performance, future urban environments, urban competitiveness, and resilience. But several fundamental architectural components must be in place to make a city smart.

Essential components. The basic underpinnings of a smart city include five components:

Broadband infrastructure. This infrastructure is pivotal, offering connectivity to citizens, institutions, and organizations. However, today's Internet lacks the robustness needed to support smart cities' services and data volume. It includes both wired and wireless

networks. Wireless broadband is important for smart cities, especially with the explosive growth of mobile applications and popularity and the connectivity of smart devices;

E-services. The concept of "electronic services" involves using ICT in the provision of services, including sales, customer service, and delivery. The Internet is today the most important way to provide them (such as for tourism, city environment, energy, transport, security, education, and health). A European Union research initiative (called the innovation framework H2020) focuses on developing such e-services; and

Open government data. Open government data (OGD) means data can be used freely, reused, and redistributed by anyone. A multinational initiative to promote worldwide adoption of OGD was launched in 2012 with input from the Microsoft Open Data initiative, Organization for Economic Cooperation and Development, and U.S. Open Data Initiative (http://www.data.gov).

A smart city can be seen as an open data generator. Implementation questions include: How can we efficiently sort and filter the data being produced? Who are the legal owners of the data? And what are the restrictions on the data? Navigating OGD can be related to the open data barriers of task complexity and information quality. With OGD, there is no explanation of the data's meaning, and discovering the appropriate data is difficult. Data is often duplicated, and data formats and datasets are likewise often too complex for humans and even machines to handle. In 2015, the Spanish standardization normative UNE178301 was published to help cities evaluate the maturity of their own open data projects by relying on five characteristics of their data—political, legal, organizational, technical, and social. OGD will have an enormously positive effect on services offered to citizens, thus improving their daily lives. In this context, linked open data (LOD) could support complex and interdisciplinary data mining analysis.[12] LOD is complex because data viewed in isolation may be irrelevant but when aggregated from various sources can yield more meaningful results and fresh insights. LOD is used in such applications as linked data in libraries for creating globally interlinked library data, linked data in biomedicine for creating orthogonal interoperable reference ontologies, and linked government data for improving internal administrative processes.

By leveraging advanced power systems, networking, and communication technologies, a smart city aims to enhance the lives of its citizens and optimize territorial, economic, and environmental resources.

Sustainable infrastructures. The International Electrotechnical Commission (IEC) says cities aiming to develop into smart cities should start with three pillars of sustainability: economic, social, and environmental. One of the first steps in addressing sustainability is to increase resource efficiency in all domains (such as energy, transport, and ICT). An efficient and sustainable ICT infrastructure is essential for managing urban systems development. Adepetu et al.[1] explained how an ICT model works and can be used in sustainable city planning. For a sustainable ICT infrastructure, they defined various green performance indicators for ICT resource use, application lifecycle, energy impact, and organizing impact.

E-governance. This component focuses on a government's performance through the electronic medium to facilitate an efficient, speedy, transparent process for disseminating information to the public and also for performing administration activities. An e-government system consists of three components: government-to-citizen, government-to-business, and government-to-government. E-government allows citizens to fulfill their civic and social responsibilities through a Web portal. A growing number of governments around the world are deploying Web 2.0 technologies, an architecture referred to as "e-government 2.0," linking citizens, businesses, and government institutions in a seamless network of resources, capabilities, and information exchange.

Fundamental technologies. The design and implementation of smart cities also involves a number of technologies:

Ubiquitous computing. Ubiquitous devices include heterogeneous ones that communicate directly through heterogeneous networks. The UrBan Interactions Research Program[d] at the University of Oulu, Oulu, Finland, studies urban computing, interaction among urban spaces, humans,

d Open Ubiquitous Oulu; http://www.ubioulu.fi/en/home

information technology, and information.[8,18] Ferreira[8] described how to build context-aware applications, collect data, and study human behavior. To support a smart and ubiquitous environment, telecommunication infrastructures, Lee[13] said, should be enhanced to provide a better understanding of networks, services, users, and users' devices with various access connections. Lee[13] also identified six capabilities and functions of smart ubiquitous networks including context awareness, content awareness, programmability, smart resource management, autonomic network management, and ubiquity.

Big data. Traditional database management tools and data processing applications cannot process such a huge amount of information. Data from multiple sources (such as email messages, video, and text) are distributed in different systems. Copying all of it from each system to a centralized location for processing is impractical for performance reasons. In addition, the data is unstructured. Deploying thousands of sensors and devices in a city poses significant challenges in managing, processing, and interpreting the big data they generate. Big data,[10] reflecting such properties as volume, variety, and velocity, is a broad term for complex quantitative data that requires advanced tools and techniques for analyzing and extracting relevant information. Several challenges must be addressed, including capture, storage, search, processing, analysis, and visualization. Also needed is a scalable analytics infrastructure to store, manage, and analyze large volumes of unstructured data. Smart cities can use multiple hardware and software technologies to process the big data being produced, including parallel system architectures (such as cluster-based high-performance systems and cloud platforms), parallel file systems and parallel input/output (such as parallel file systems for big data storage and NoSQL databases for big data), programming (such as low-level programming models, skeletal parallel programming, and generic parallel programming), data management on multilevel memory, and hierarchy task scheduling.[15]

Networking. Networking technologies enable devices and people to have reliable communications with one another. Several wireless networking technologies, including radio frequency identification (RFID), ZigBee, and Bluetooth, have been deployed, although they are limited by the number of devices they can support, along with their throughput and transmission range. New wireless technologies (such as WiMAX and Long-Term Evolution) are unsuitable due to their high energy consumption. Novel Wi-Fi technology (such as by the IEEE 802.11ah Task Group) could be an efficient solution for smart city services.[11] IEEE 802.11ah aims to help design an energy-efficient protocol allowing thousands of indoor and outdoor devices to work in the same area and a transmission range up to 1km at default transmission power of 200mW.[11]

Smart cities need citizens to be continuously connected—in public places, in public transportation, and at home—in order to share their knowledge and experience.

IoT. One of the main IoT goals is to make the Internet more immersive and pervasive. As a network of highly connected devices, IoT technology works for a range of heterogeneous devices (such as sensors, RFID tags, and smartphones). Multiple forms of communications are possible among such "things" and devices. IoTs must be designed to support a smart city's vision in terms of size, capability, and functionality, including noise monitoring, traffic congestion, city energy consumption, smart parking meters and regulations, smart lighting, automation, and the salubrity of public buildings.[11] They must exploit the most advanced communication technologies, thus supporting added-value services for a city's administration and citizens.

Cloud computing. Cloud computing enables network access to shared, configurable, reliable computing resources. The cloud is considered a resource environment that is dynamically configured to bring together testbeds, applets, and services in specific instances where people's social interaction would call for such services;

Service-oriented architectures (SOAs). An SoA is a principle for software structuring based on service. A smart city's development should focus on SOA-based design architectures to address its challenges. A smart city thus requires a new IT infrastructure, from both a technical and an organizational perspective.

Table 1. Smart city projects around the world.

Project/location	Funding	Duration	Goals	Smart city characteristics	Partners
Yokohama Smart City Project,[a] Japan	Ministry of Economy, Trade and Industry (METI)	2010–2015	Low-carbon city, hierarchical energy management systems (EMS), sensitive photovoltaic (PV) generation	Smart environment, smart living	Tokyo Institute of Technology, Toshiba, Mitsubishi, Hitachi
Smart Mobility & Energy Life in Toyota City,[b] Japan	METI	2010–2015	PV generation, intelligent transportation systems, hierarchical EMS, 61.2% renewable energy, 4,000 next-generation vehicles	Smart mobility, smart environment	Nagoya University, Toyota City, Fujitsu, Hitachi, Toyota Motor Corporation, Chubu Electric Power Co.
Keihanna Eco City Next-Generation Energy and Social Systems project,[c] Japan	METI	2010–2015	Develop community EMS to minimize CO_2 emissions, vehicle-to-infrastructure, and to-vehicle	Smart environment	Kyoto, Kizugawa, Kyotanabe, Fuji Electric, Kyoto Center for Climate Actions, Mitsubishi
Kitakyushu Smart Community Project,[d] Japan	METI	2010–2015	Participation by citizens and companies in the energy-distribution process, PV generation, establishing charging infrastructure, and next-generation traffic systems (bicycles and public transport)	Smart mobility, smart environment	Toyota Motor Corporation, IBM Japan, Japan Telecom Information Service Corporation, Mitsubishi Heavy Industries
CITYKEYS,[e] European Union	H2020 project, European Union	2015–2017	Develop and validate key performance indicators and data-collection procedures for smart cities, sharing best practices on user privacy and other legislative issues among cities	Smart mobility, smart environment, smart living, smart people	Research organizations: VTT (Finland), AIT (Austria), TNO (The Netherlands); and five partner cities: Rotterdam, Tampere, Vienna, Zagreb, Zaragoza
LIVE Singapore project,[f] Singapore	National Research Foundation of Singapore	2011–2016	Develop open platform for collecting, elaborating, and distributing real-time data reflecting urban activities: tracking vehicular traffic and estimated temperature rise, energy consumption, and taxi operations	Smart living, smart people	MIT's SENSEable City Lab, Future Urban Mobility research initiative, Changi Airport Group, ComfortDelGro, NEA, PSA, SP Services, SingTel
SmartSantander,[g] Europe	European Union	2010–2013	Deploy 20,000 sensors in Belgrade, Guildford, Lübeck, and Santander, exploiting multiple technologies to collect information on parking spaces, public transport, and automatic management of light; currently uses 2,000 IEEE 802.15.4 devices	Smart living, smart environment	Telefonica I+D (Spain) Universität zu Lübeck (Germany), Ericsson (Serbia), Alcatel-Lucent (Italy), Alexandra Instituttet A/S (Denmark)
Open Cities project,[h] Europe	European Union	2011–2013	Explore how to implement open and user-driven innovation methodologies in the public sector in European cities. including Amsterdam, Barcelona, Berlin, Bologna, Helsinki, Paris, and Rome	Smart governance	Fraunhofer Institute FOKUS (Germany), ATOS (Spain), ESADE Business School (Spain), Berlin Government Senate Department for Economics, Technology and Women's Issues (Germany), Institut Telecom (France), NESTA (U.K.)
Vehicle2Grid,[i] The Netherlands	Top consortium on Knowledge and Innovation Switch2SmartGrids	2014–2017	Deliver European Open Data repository	Smart mobility, smart environment	Cofely, Alliander, ABB, Mitsubishi Motors Corporation, Amsterdam Smart City, Amsterdam University of Applied Sciences
City Science Initiative,[j] MIT/U.S.	Corporate sponsorship, industrial funding, National Science Foundation, Defense Advanced Research Projects Agency, National Institutes of Health	—	Use batteries in electric cars to store locally produced energy	Smart mobility, smart environment	27 scientific research teams[k]
"Green Vision"[l] initiative, San Jose, CA	State and federal funding	2007–2022	Gain scientific understanding of cities: urban analytics, governance, mobility networks, electronic and social networks, and energy networks Create clean tech jobs, reducing energy use by 50%, generating 100% energy from renewable sources, reusing water, installing zero-emission lighting, and having 100% public vehicles run on alternative fuels	Smart mobility, smart environment	Universities, private companies, regional agencies

a http://www.city.yokohama.lg.jp/ondan/english/yscp/
b http://jscp.nepc.or.jp/article/jscpen/20150528/445244/
c http://jscp.nepc.or.jp/en/keihanna/index.shtml
d http://www.nedo.go.jp/content/100639530.pdf
e http://www.citykeys-project.eu
f http://senseable.mit.edu/livesingapore/
g http://www.smartsantander.eu/
h http://www.opencities.net/content/project
i http://amsterdamsmartcity.com/?lang=en
j http://cities.media.mit.edu/
k http://www.media.mit.edu/research/groups-projects
l http://www.sanjoseca.gov/DocumentCenter/View/42557

Cybersecurity architectures. Smart cities pose challenges to the security and privacy of citizens and government alike. The security issues associated with the information produced in a smart city extend to relationships among those citizens, as well as their personal safety. Some smart cities are already confronted by identity spoofing, data tampering, eavesdropping, malicious code, and lack of e-services availability. Other related challenges include scalability, mobility, deployment, interoperability (of multiple technologies), legal, resources, and latency. Critical infrastructure needs protection from attacks that could cripple or severely damage a city's ability to function, from industrial plants to essential services, including access to electricity, water, and gas. Attackers exploiting vulnerabilities in industrial control systems (such as the supervisory control and data acquisition, or SCADA, systems that manage it) can cause significant disruption in service delivery. In vehicular ad hoc networks (VANETs), the standard 1609.2-2013 recommends using pseudonymous authentication for protecting the location privacy of vehicles. Adding layers of security (such as VANETs, SCADA, and mobile networks) in an independent way will make the architecture of a smart city very complex. A smart city thus requires a cybersecurity architecture by design.

Projects and Standardization Efforts

Many cities have piloted their own smart city projects; see Table 1 for specific smart city developments in Asia, Europe, and North America.

Smart cities incorporate digital ser-

Table 2. Smart city benchmarking tools.

Reference	Benchmarking tool	Description	Type of comparison
Pires et al.[19]	Local sustainable development indicators	Calculation of a single percentage synthesizing multiple themes: environmental sustainable development education, energy, water, transport, noise, agriculture, tourism, nature conservation, marine and coastal environment, and biodiversity.	Quantitative
IBM*	IBM smarter city assessment tool	A tool to measure a city's performance, perform benchmarks, and identify challenges and opportunities for improvement; tool relies on multiple themes: civic life, economic life, mobility and transport, energy management, water management, and city services.	Qualitative, quantitative
Lee et al.[14]	Framework for building smart cities	Taking various practical perspectives from case studies in San Francisco and Seoul Metropolitan City, the framework relies on urban openness, service innovation, partnership formation, urban proactiveness, smart city infrastructure integration, and smart city governance.	Qualitative
Desouza et al.[6]	Resilience framework for cities	An approach to designing, planning, and managing for resilience, including evaluation of cultural and process dynamics within cities.	Qualitative
Debnath et al.[5]	Benchmarking for smart-transport cities	A composite scoring system to measure the "smartness index" of a city's transportation system; proposed indicators rely on private transport (traffic flow prediction, parking information sharing, paying tolls/parking charges/enforcement fines, and automated and coordinated traffic signal control), public transport (detection of passengers, passenger information management, and detection of passengers), and emergency transport (with emergency vehicles able to provide a priority signal).	Quantitative

* http://www.ibm.com/smarterplanet/us/en/smarter_cities/solutions/solution/S868511G94528M58.html

Table 3. Related work by various standardization organizations.

Organization	Corresponding work related to smart cities	Description
International Electrotechnical Commission	Electro-technical field, power-utility automation, energy management, distribution management	Plan and organize standardization activities in area under consideration (Systems Evaluation Group)
International Telecommunication Union	Environment and climate change*	Create framework for standards on smart and sustainable cities (ITU-T Study Group 5)
IEEE	Smart grid, IoT, intelligent transportation systems, eHealth, and smart interoperability	IEEE 2030 covers smart interoperability, including characteristics, performance, and evaluation criteria; IEEE 1901 covers smart grid networks, including coexistence of broadband power line systems operating on same power line; additional standards (IEEE 1609.2) cover intelligent transportation systems.
European Telecommunications Standards Institute	Smart and sustainable cities	Establish coordination group for standardization of smart and sustainable cities.
European Commission	Energy transport and information and communication technology	Create European Innovation Partnership on Smart Cities and Communities and European Smart Cities Ranking project.**
Britsh Standards Institution	Definition of standardization strategy	Develop a guide for strategies for smart cities and communities, a smart city data concept model, and collaborative relationship management.

* http://www.itu.int/en/ITU-T/studygroups/2013-2016/05/Pages/default.aspx
** http://www.smart-cities.eu/?cid=-1&ver=3

vices as part of a livable and sustainable environment for their citizens. Along with their benefits comes a set of new issues regarding, say, open data and interoperability for policymakers, companies, and citizens alike. Cities need performance-evaluation indicators to measure how much they might possibly improve quality of life and sustainability. Performance evaluation indicators often lack standardization, consistency, or comparability from city to city. A series of international standards is being developed to provide a holistic, integrated approach to sustainable development and resilience under ISO/TC 268 (sustainable cities and communities standard).

ISO 37120 establishes a set of standard performance-evaluation indicators that provide a uniform approach to what is measured and how that measurement is to be undertaken. The International Telecommunications Union defines various key performance indicators for smart sustainable cities through the standard ISO/TR 37150 for fire and emergency response, health, education, safety, transportation, energy (the percentage of a city's population with authorized electrical service), water (the percentage of a city's population with a potable water supply service), social equity, technology and innovation (such as number of Internet connections per 100,000 people), CO_2 levels and reduction strategies, and buildings (such as energy consumption of residential buildings). Fujitsu has also proposed a performance indicator for ICT in smart cities, using such factors as a city's environmental impact, the ratio of renewable energy to total energy consumed, and a community's power-outage frequency rate.

Some type of benchmarking method is pertinent to compare smart cities. Various benchmarking methods were proposed in 2014 for such comparison. For instance, Pires et al.[19] analyzed a Portuguese initiative that uses common indicators to benchmark sustainable development across 25 Portuguese cities and municipalities.[e] The Smarter City assessment tool developed by IBM

Detecting behavioral anomalies in daily human life is very important for developing smart systems.

can be used to measure a city's performance against indicators for each smart city system. It identifies opportunities for improvement by evaluating component effectiveness, taking into account their degree of modernity, CO_2 emissions, and the city's energy management. This analysis is based on multiple indicators to measure the current situation and the potential for improving a city for the following components of civil society: civic life (degree of modernity, public safety, education, and housing); economic life (business attraction strategy and online services for businesses); mobility and transport (wireless networks and telecommunications infrastructures); energy management (production rate of renewable energy, smart metering, and CO_2 emissions); water management (water quality and smart metering); and city services. Another smart city tool, developed by Boyd Cohen[f] of Universidad del Desarrollo in Chile, includes the environment, mobility, governance, economy, people, and living components (such as health, safety, cultural level, and happiness).

Table 2 outlines the benchmarking methods aimed at measuring smart cities from multiple perspectives.

Various organizations and academic institutions have been working on smart city models. As mentioned earlier, the IEC uses three pillars of sustainability—economic, social, and environmental—to develop smart cities. IBM combines instrumentation, interconnection, and intelligence in its smart cities model. Table 3 outlines some of the efforts being undertaken by various organizations.

Challenges and Research Opportunities

Here, we highlight some of the challenges faced by smart cities while exploring research opportunities that need more attention to assist smart city development and adoption.

Challenges. The following are the most noteworthy challenges to be addressed.

Lack of investment. The concept of smart cities reflects strong potential for investment and business opportunities.

e Smart Cities Portugal; http://www.inteli.pt/uploads/documentos/documento_1400235009_2055.pdf

f http://smartcitiescouncil.com/resources/smart-city-index-master-indicators-survey

Figure 2. Smart city infrastructure investments by industry, 2014–2023.

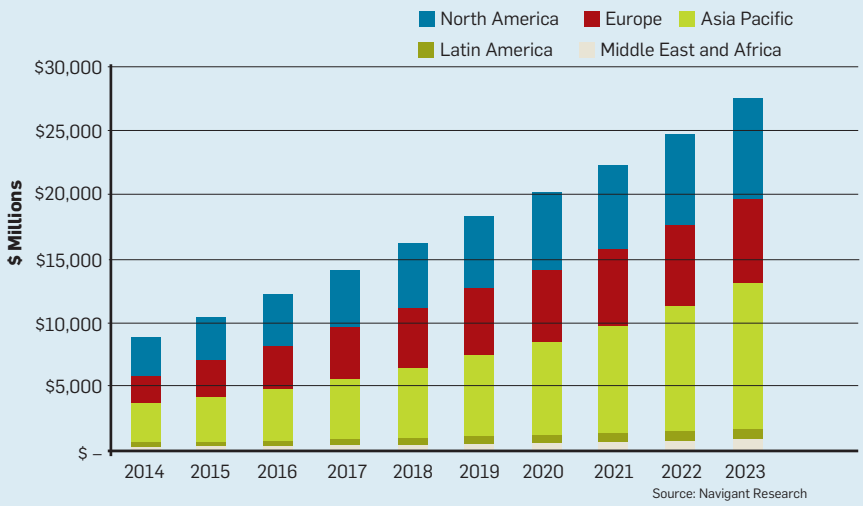
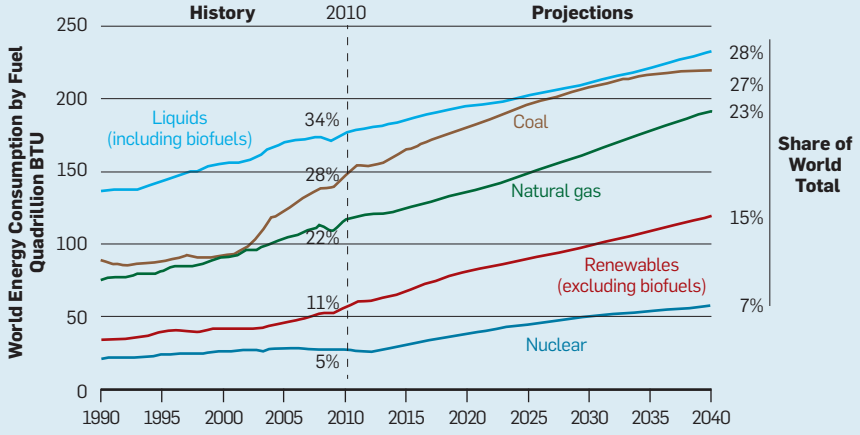


Figure 3. Estimating future energy consumption.



International Energy Outlook 2013; http://www.eia.gov/forecasts/archive/ieo13/pdf/0484%282013%29.pdf

On the one hand, investment in related projects has grown in recent years, financed by both governments (including municipalities and public research agencies) and private entities (companies and citizens). Navigant Research says investment in smart cities is divided into smart government, smart building, smart transport, and smart utilities. By 2020, $13 billion in funding is expected to establish smart cities all over the world (see Figure 2). Yet according to a 2014 research report on financing models for smart cities, Navigant Research said this infrastructure faces major financial hurdles, including the perceived high risk of investing in innovative solutions, uncertainty of energy price policies, major investment required, long-term delays before reaping profits, and limited capacity for public funding.

Cost. Many cities are committing large budgets to get smarter. For instance, Saudi Arabia is investing U.S.$70 billion in the King Abdullah Economic City (2005–2020) in collaboration with Orange Business Services, Ericsson, Siemens, Cisco, and other companies. In Dubai, 1,000 government services are expected to be "smart" in a few years. In 2015, South Africa began a $7.4 billion smart city project[g] to achieve carbon neutrality in public transport by 2025. The bus system in Copenhagen, Denmark, costs €125 million annually. In India, the national government's annual budget for development of 100 smart cities[h] is $1.27 billion, adding 11.5 million homes annually. In the European Union, smart city market projections are expected to exceed $1 trillion by the end of 2016. China's future smart cities allocations exceed $322 billion for more than 600 cities nationwide.[i] All these projects demonstrate how substantial is the rate of investment in smart cities. However, if some of the challenges (such as cybersecurity) are not addressed early, the ultimate cost of smart cities will only increase.

High energy consumption. The U.S. Energy Information Administration estimates approximately 21% of the world's electricity generation was from renewable energy in 2011, with a projected increase to nearly 25% by 2040. The absence of natural resources in the estimation of energy consumption for the rest of the 21st century plays a negative role in smart cities investments (see Figure 3). The future of energy cost and access is uncertain due primarily to their dependence on projected geopolitical, socioeconomic, and demographic scenarios.

Smart citizens. Social dimensions must also be taken into consideration. A city's "smartness" greatly depends on citizens' participation in smart city projects, through multiple communication tools (such as a municipality's Web portal, social networks, and smartphone applications). Smart cities need citizens to be continuously connected—in public places, in public transportation, and at home—in order to share their knowledge and experience. The objective is effective management of natural resources and a higher quality of life for citizens; for example, they can compare their household use of electricity, gas, and water through their smartphones. Adequately maintaining this social dimension is a challenge, though it is a vital aspect of a smart

g http://www.africapropertynews.com/southern-africa/3071-construction-begins-on-south-africa-7-4bn-smart-city.html

h http://smartcitiescouncil.com/article/india-budgets-smart-infrastructure

i https://eu-smartcities.eu/blog/eib-eip-smart-cities-financing

city's functionality that, exploited correctly, yields dividends for both citizens and the city.

Privacy. Privacy will play a pivotal role in any smart city strategy. Citizens interact with smart city services through their smartphones and computers connected through heterogeneous networks and systems. It is thus imperative smart cities, founded on the use of ICT, be adept at handling important privacy issues (such as eavesdropping and confidentiality). Domingo-Ferrer[7] divided privacy into three dimensions: respondent, user, and owner. Martinez-Ballesté et al.[16] proposed the concept of citizens' privacy based on statistical disclosure control, including methods for safeguarding the confidentiality of information about individuals when releasing their data. Examples of such methods are private information retrieval (obtaining database information belonging to someone and hiding it), privacy-preserving data mining (collaboration between entities to get results without sharing all the data), location privacy, anonymity and pseudonyms, privacy in RFID, and privacy in video surveillance.

Cyberattacks. As with any infrastructure, smart cities are prone to cyberattack, and the current attack surface for cities is wide open. IOActive Labs identified several causes of cyberattack: lack of cybersecurity testing, poor or nonexistent security features in connected devices, poor implementation of security features, encryption (outdated and weak encryption algorithms), lack of computer emergency response teams, large and complex attack surfaces, patch deployment issues, insecure legacy systems, lack of cyberattack emergency plans, and denial of service (DoS).

A 2015 workshop[j] identified several challenges, including vulnerabilities in the transfer of data, physical consequences for cyberattacks, collection and storage of large amounts of data in the cloud, and exploitation of city data by attackers.

Detecting behavioral anomalies in daily human life is very important for developing smart systems. Table 4 outlines some research anomaly-detection frameworks being proposed to detect behavioral anomalies in human daily life in smart-home and smart-grid environments in the context of smart cities. Multiple challenges include characterization of the behaviors of sensor nodes (such as the accuracy of detection, false positive rate, and low computational cost).

Research opportunities. Consider these research opportunities.

IoT management. The IoT needs an efficient, secure architecture that enhances urban data harvesting. As others have noted, ubiquitous and collaborative urban sensing integrated with smart objects can provide an intelligent environment. Otherwise, packet latencies and packet loss are inevitably not controllable. One such proposal is the Mobile Ad hoc Networks (MANET) coordination protocol to opportunistically exploit MANET nodes as mobile relays for the fast collection of urgent data from wireless sensor networks without sacrificing battery lifetime. Simulation results show that their cluster formation protocol is reliable and always delivers over 98% of packets in street and square scenarios. Other issues, including the convergence of IoT and intelligent transportation systems require further investigation.

Data management. Data plays a key role in a smart city. A huge quantity of data will be generated by smart cities; understanding, handling, and treating it will be a challenge. However, mobile phone data can help achieve several smart city objectives. Smartphone data can be used to develop a variety of urban applications. For example, transportation analysis through mobile phone data can be applied for estimating road traffic volume and transport demands. Real-time information from mobile-phone data about the origins of visitors combined with taxis' Global Positioning System data could help manage transportation resources, as in, say, the public's future demand for taxis.

j Designed-In Cybersecurity for Smart Cities Workshop, May 2015; http://www.nist.gov/cps/cybersec_smartcities.cfm

Table 4. Smart home and smart grid intrusion-detection systems.

Approach	Attacks	Detection method	Advantages	Disadvantages
Zhu et al.[24]	Spatial anomaly (sleeping at the wrong place), timing anomaly (sleepwalking at midnight); duration anomaly (working on the computer for a long time); sequence anomaly (working on the computer for a very long time without eating)	Dynamic Bayesian network; maximum-likelihood estimation algorithm; and Laplace smoothing	Accuracy of 99%, low false positive rate	User intervention needed
Usman et al.[23]	Transmission errors, node faults, or attacks	Fuzzy logic-based cross-layer rule-base	Accuracy of 98%, low energy consumption, simplicity	Tested on limited profiles of sensor nodes
Mitchell et al.[17]	Attacks performed by a compromized device in smart grid	Behavior-rule based intrusion detection system	High intrusion detection rate, false positives less than 6%	No repair strategies
Britsh Standards Institution	Definition of standardization strategy	Britsh Standards Institution	Definition of standardization strategy	Develop a guide for strategies for smart cities and communities, a smart city data concept model, and collaborative relationship management

A smart city also needs a large amount of data storage, which can be achieved through, say, an electrically tunable metasurface, or an array of nano-antennas.[21,22] This technology could affect a range of fields (such as imaging, communication, encryption, and data storage).[9] The response of a given smart city to uncertain data is significant, leading to the question of how to measure the robustness of a smart city.

Smart city assessment framework. Livable cities must be monitored through a quality-of-living index, measuring the health, safety, and prosperity in the city. An assessment framework must take into consideration various characteristics, including smart city strategy and the interests of all stakeholders (such as performance evaluators, ICT infrastructures, legal and regulatory policies, services, business models, and sustainability). The objectives of such a framework are to compare the characteristics of different smart cities to identify new challenges, quantify benefits, and evaluate performance.

VANET security. In smart cities, efficient security support is an important requirement of VANETs. One consideration is how to secure them by designing solutions that reduce the likelihood of network attacks or even how to diminish the effect a successful attack could have on them.

Several security challenges persist in the realm of authentication and driver-behavior analysis. A smart city needs lightweight, scalable authentication frameworks that protect drivers from internal and external attackers. The IEEE 1609.2 v2 standard specifies a set of security services (such as certificate authority) for supporting vehicular communications. However, public key infrastructure-based (PKI-based) solutions to verify vehicle authenticity might not lead to a scalable solution. Cross-certification should be defined when countries or cities have multiple root certificate authorities. The authority responsible for misbehavior detection should be incorporated into the PKI system. City governments need to investigate innovative fast, low-cost message-exchange solutions whose communication overheads remain constant as the number of vehicles within communication range increases. On the other hand, to ensure vehicle privacy and anonymity, pseudonym change (pseudonymous certificates with removed identifiable information are used in PKI) strategies (such as fixed time change, random change, and density-based change) should be designed and tested on a large scale, as in a smart city environment. Desired security properties include lightweight authentication, security architecture scalability, and resilience against malicious nodes and DoS attacks.

Elliptic curve cryptography is considered the most trusted solution for providing security on resource-constrained devices and embedded systems.

Improving photovoltaic cells. Achieving a sustainable energy source in a smart city must include renewable energy. Solar technology has made significant strides (such as photovoltaic cells that convert light energy into electricity) in the past decade. Still, more efficient solar-energy-harvesting techniques are needed to improve solar cells. The vast majority of today's solar cells absorb light through a thin film of cadmium telluride composed of two layers of glass. The benefits of thin-film solar cells include low production cost and ease of fabrication. New design approaches emerged in 2014 based on nanophotonics, whereby nano-antennas are exploited for guiding and localizing light at the nanoscale. Such designs promise to improve the absorption process in photovoltaic devices, thus enabling a considerable reduction in the physical thickness of solar-absorber layers and paving the way for new solar-cell designs. In this context, designing new micro- and nano-antennas for improving light absorption is important, and dimer nanostructures, or structures composed of two nanoparticles turn out to be good candidates.[2]

Smart city enablers. Technology advances are helping create a market for smart city products and solutions, but smart cities need efficiency and sustainability. Angelidou[4] proposed the conjuncture of four forces: urban futures, technologies, applications, and innovation economy. To develop, smart cities must leverage technological advancements and development of knowledge and innovation networks.[4] In addition, to create a smart city, Amaba[3] pro-

posed integrating multimedia, human factors, user-centered system methodology, and design principles. Smart cities can provide solutions to many sustainability problems, but their cohesive development requires an effective policy to be in place as part of any solution.

Information system risks. In a smart city, everything is interconnected, including the public water system, traffic control, public transportation, and critical infrastructure. Each one involves its own vulnerabilities. Although a smart city is a complex system, its interconnected nature means a single vulnerability could greatly affect citizens' security; for example, an attacker might be able to connect to the electric power system to gain access to the network and alter public transportation to potentially paralyze intelligent transportation systems, with thousands of passengers on board at rush hour. The attacker could also launch false alarms and modify traffic lights and controllers. Finding workable solutions is critical; otherwise the public will not trust smart city projects or deem them viable. Devices (such as smartphones) used in the IoT depend on security features (such as the ability to do secure email, secure Web browsing, and other transactions). These devices need efficient, secure implementations of all these features in a smart city. In this context, elliptic curve cryptography (ECC) is considered the most trusted solution for providing security on resource-constrained devices and embedded systems. ECC uses public key cryptography and is based on the algebraic structure of elliptic curves over finite fields. ECC advantages include the ability to provide the same level of security as other cryptographic algorithms but with smaller keys, low memory requirements and computational overheads, and much faster computations.

The scientific community must address these cybersecurity projects. Unresolved challenges and opportunities for participation involve DoS-attack detection for distribution systems, cryptographic countermeasures, and authentication in IoT and critical infrastructure, as well as in key management.

Conclusion

The strong interest by municipal and local governments worldwide in smart cities stems from their ability to improve their citizens' quality of life. Here, we described some of the basic concepts of smart cities, identifying challenges and future research opportunities to enable large-scale deployment of smart cities. Developers, architects, and designers should now focus on aspects of IoT management, data management, smart city assessment, VANET security, and renewable technologies (such as solar power). We underscore when designing smart cities, security and privacy remain considerable challenges that demand proactive solutions. We hope to see smart city developers, architects, and designers provide scalable, cost-effective solutions to address them in the future.

Acknowledgments

We express our gratitude to Teodora Sanislav, Nils Walravens, and Lyes Khoukhi for their comments and suggestions on early drafts of this article, helping improve its content, quality, and presentation. We would also like to thank the reviewers for their valuable feedback and reviews. C

References

1. Adepetu, A., Arnautovic, E., Svetinovic, D., and de Weck, L. Complex urban systems ICT infrastructure modeling: A sustainable city case study. *IEEE Transactions on Systems, Man, and Cybernetics: Systems 44*, 3 (Mar. 2014), 363–374.
2. Akselrod, G., Argyropoulos, C., Hoang, T., Ciracì, C., Fang, C., and Huang, J. Probing the mechanisms of large Purcell enhancement in plasmonic nanoantennas. *Nature Photonics 8* (2014), 835–840.
3. Amaba, B. Industrial and business systems for smart cities. In *Proceedings of the First International Workshop on Emerging Multimedia Applications and Services for Smart Cities.* ACM Press, New York, 2014, 21–22.
4. Angelidou, M. Smart cities: A conjuncture of four forces. *Cities 47* (Sept. 2015), 95–106.
5. Debnath, A., Chin, H., Haque, M., and Yue, B. A methodological framework for benchmarking smart transport cities. *Cities 37* (Apr. 2014), 47–56.
6. Desouza, K. and Flanery, T. Designing, planning, and managing resilient cities: A conceptual framework. *Cities 35* (Dec. 2013), 89–99.
7. Domingo-Ferrer, J. A three-dimensional conceptual framework for database privacy. Chapter in *Secure Data Management*, W. Jonker and M. Petkovic, Eds. Springer, Berlin, Heidelberg, Germany, 2007, 193–202.
8. Ferreira, D. *AWARE: A Mobile Context Instrumentation Middleware to Collaboratively Understand Human Behavior.* Ph.D. dissertation, Faculty of Technology, University of Oulu, Oulu, Finland, 2013; http://jultika.oulu.fi/Record/isbn978-952-62-0190-0
9. Huang, L., Chen, X., Mühlenbernd, H., Zhang, H., Chen, S., Bai, B., Tan, Q., Jin, G., Cheah, K., Qiu, C., Li, J., Zentgraf, T., and Zhang, S. Three-dimensional optical holography using a plasmonic metasurface. *Nature Communications 4*, 2808 (Nov. 2013).
10. Jagadish, H.V., Gehrke, J., Labrinidis, A., Papakonstantinou, Y., Patel, J., Ramakrishnan, R., and Shahabi, C. Big data and its technical challenges. *Commun. ACM 57*, 7 (July 2014), 86–94.
11. Khorov, E., Lyakhov, A., Krotov, A., and Guschin, A. A survey on IEEE 802.11ah: An enabling networking technology for smart cities. *Computer Communications 58*, 1 (Mar. 2015), 53–69.
12. Lausch, A., Schmidt, A., and Tischendorf, L. Data mining and linked open data—New perspectives for data analysis in environmental research. *Ecological Modelling 295*, 10 (Jan. 2015), 5–17.
13. Lee, C., Gyu, M., and Woo, S. Standardization and challenges of smart ubiquitous networks. *IEEE Communications Magazine 51*, 10 (Oct. 2013), 102–110.
14. Lee, J., Gong, M., and Mei-Chih Hu, H. Towards an effective framework for building smart cities: Lessons from Seoul and San Francisco. *Technological Forecasting and Social Change 89* (Nov. 2014), 80–99.
15. Ma, Y., Wu, H., Wang, L., Huang, B., Ranjan, R., Zomaya, A., and Jie, W. Remote sensing big data computing: Challenges and opportunities. *Future Generation Computer Systems 51* (Oct. 2015), 47–60.
16. Martinez-Balleste, A., Perez-Martinez, P., and Solanas, A. The pursuit of citizens' privacy: A privacy-aware smart city is possible. *IEEE Communications Magazine 51*, 6 (June 2013), 136–141.
17. Mitchell, R. and Chen, I. Behavior-rule-based intrusion detection systems for safety-critical smart grid applications. *IEEE Transactions on Smart Grid 4*, 3 (Sept. 2013), 1254–1263.
18. Perttunen, M., Riekki, J., Kostakos, V., and Ojala, T. Spatio-temporal patterns link your digital identities. Computers, *Environment and Urban Systems 47* (Sept. 2014), 58–67.
19. Pires, S., Fidélis, T., and Ramos, T. Measuring and comparing local sustainable development through common indicators: Constraints and achievements in practice. *Cities 39* (Aug. 2014), 1–9.
20. Rifkin, J. *The Zero Marginal Cost Society: The Internet of Things, The Collaborative Commons, and The Eclipse of Capitalism.* St. Martin's Press, St. Martin's Griffin, New York, 2015.
21. Sheldon, M., Van de Groep, J., Brown, A., Polman, A., and Atwater, H. Plasmoelectric potentials in metal nanostructures. *Science 346*, 828 (Nov. 2014), 828–831.
22. Silva, A., Monticone, F., Castaldi, G., Galdi, V., Alù, A., and Engheta, N. Performing mathematical operations with metamaterials. *Science 343*, 6167 (Jan. 2014), 160–163.
23. Usman, M., Muthukkumarasamy, V., and Wu, X. Mobile agent-based cross-layer anomaly detection in smart home sensor networks using fuzzy logic. *IEEE Transactions on Consumer Electronics 61*, 2 (May 2015), 197–205.
24. Zhu, C., Sheng, W., and Liu, M. Wearable sensor-based behavioral anomaly detection in smart assisted-living systems. *IEEE Transactions on Automation Science and Engineering 12*, 4 (Oct. 2015), 1225–1234.

Rida Khatoun (rida.khatoun@telecom-paristech.fr) is an associate professor at Telecom ParisTech, Paris, France.

Sherali Zeadally (szeadally@uky.edu) is an associate professor in the College of Communication and Information at the University of Kentucky, Lexington, KY.

DOI:10.1145/2964342

John H. Holland's general theories of adaptive processes apply across biological, cognitive, social, and computational systems.

BY STEPHANIE FORREST AND MELANIE MITCHELL

Adaptive Computation: The Multidisciplinary Legacy of John H. Holland

IN AUGUST 2015, Professor John H. Holland passed away in Ann Arbor, MI, where he had served on the University of Michigan faculty for more than 50 years. John, as he was known universally to his colleagues and students, leaves behind a long legacy of intellectual achievements.

As a descendant of the cybernetics era, he was influenced by the work of John von Neumann, Norbert Wiener, W. Ross Ashby, and Alan Turing, all of whom viewed computation as a broad, interdisciplinary enterprise. Holland thus became an early proponent of interdisciplinary approaches to computer science and an active evangelist of what is now called computational thinking, reaching out enthusiastically to psychologists, economists, physicists, linguists, philosophers, and pretty much anyone he came in contact with. As a result, even though he received what was arguably one of the world's first computer science Ph.D. degrees in 1959,[23] his contributions are sometimes better known outside computer science than within.

Holland is best known for his invention of genetic algorithms (GAs), a family of search and optimization methods inspired by biological evolution. Since their invention in the 1960s, GAs have inspired many related methods and led to the thriving field of evolutionary computation, with widespread scientific and commercial applications. Although the mechanisms and applications of GAs are well known, they were only one offshoot of Holland's broader motivation—to develop a general theory of adaptation in complex systems.

Here, we consider this larger framework, sketching the recurring themes that were central to Holland's theory of adaptive systems: discovery and dynamics in adaptive search; internal models and prediction; exploratory modeling; and universal properties of complex adaptive systems.

Discovery and Dynamics in Adaptive Search

Holland's goal of developing a general theory of adaptation was spurred by both his early work on computer models of Hebbian learning[25] and his reading of Ronald Fisher's classic book, *The Genetical Theory of Natural Selection*, which integrated genetics with Darwinian selection.[8] As he read further in evolutionary biology, economics, game theory, and control theory, Holland recognized that adaptation is central to

» key insights

- **Adaptation is an open-ended dynamic process involving populations of agents operating in perpetually novel environments, continually changing their behavior to improve performance and enhance their chance of survival.**
- **Adaptive systems create, update, and use internal models of their environment to make predictions; successful models create valid homomorphisms of their environment.**
- **Exploratory models of complex adaptive systems lead to insights about essential mechanisms and principles, a use of modeling that is different from simply making accurate predictions.**

PHOTO COURTESY OF SANTA FE INSTITUTE

all of these fields; they all concern populations of agents that must continually obtain information from uncertain, changing environments and use it to improve performance and enhance the chance of survival.

Holland also recognized that adaptation must be continual and open ended. In his view, adaptive systems never achieve a state of equilibrium or final "optimum" configuration. This emphasis on open-ended, non-equilibrium dynamics was in stark contrast with the mainstream approach (at the time) in all these fields—the belief that solving for stable equilibrium dynamics was the scientific goal. Holland's contrary view was that a system in stable equilibrium is essentially dead.

Underlying Holland's theory of adaptation are the following core ideas:

Populations, sampling, and implicit parallelism. Evolution is a form of search that leverages statistics to direct population dynamics. Initially, the population is an independent sample of many individuals (from the space of all possible individuals) and over time the sample is increasingly biased toward the high-fitness regions of the search space. In addition, the population can be viewed as an implicit sample of the much larger space of traits exhibited by those individuals. Holland termed this implicit large-scale sampling of traits "implicit parallelism." Evolutionary dynamics biases these samples over time toward high-fitness regions of the search space.

Building blocks and recombination. In a population undergoing adaptation, individuals can be decomposed into building blocks—sets of traits that are the evolutionary "atoms" of an individual's fitness or performance. (As an example from biology, Holland often mentioned the Krebs cycle, a core cellular metabolic pathway that is highly conserved across living systems.) Successful individuals are discovered in stages, first by finding useful building blocks through stochastic sampling, and over time recombining them to create higher-fitness individuals out of more complex building blocks.

Exploitation vs. exploration. Successful adaptation requires maintaining a balance between exploitation, in which successful building blocks propagate in a population, and exploration, in which existing building blocks are recombined or mutated in new ways.

Inspired by Bellman[3] and others, Holland formalized the exploitation-vs.-exploration trade-off as an idealized "two-armed bandit" problem. Given a slot machine with two arms, each of which has an unknown payoff probability, how should you allocate trials (pulls) between the arms so as to maximize your total payoff? Holland argued that the optimal strategy allocates trials to the observed best arm at a slightly higher rate than an exponential function of the trials allocated to the observed worse arm.[11,12]

Further, Holland showed that in populations undergoing adaptation, building blocks are analogous to arms on a multi-armed bandit. Evaluating an individual in an environment is analogous to pulling selected arms on a multi-armed bandit, where the arms correspond to each of the building blocks making up that individual.

The question of balancing exploitation and exploration—how to optimally allocate trials to different arms based on their observed payoffs—now becomes the question of how to sample optimally in the vast space of possible building blocks, based on their

estimated contribution to fitness. Evolution deals in populations of individuals, of course, not building blocks. There is no explicit mechanism that keeps statistics on how building blocks contribute to fitness. Holland's central idea here is that nearly optimal building-block sampling occurs implicitly, as an emergent property of evolutionary population dynamics.

Holland's early papers[10,11] and his influential 1975 book *Adaptation in Natural and Artificial Systems*[12] developed a general, formal setting in which these ideas could be expressed mathematically. It was this formalization that led to the invention of genetic algorithms that featured stochastic population-based search, as well as crossover between individuals as a critical operation that allows successful building blocks to be recombined and tested in new contexts.

However, Holland's aim was more general than a new class of algorithms. He aspired to develop an interdisciplinary theory of adaptation, one that would inform, say, biology as much as computer science.[5] The later, successful application of genetic algorithms to real-world optimization and learning tasks was, for Holland, just icing on the cake. His broader view of adaptation has inspired many and engendered criticism from others, leading to spirited intellectual debates and controversies. Most controversial is the extent to which it can be demonstrated, either empirically or mathematically, that the behavior of adaptive systems is actually governed by Holland's proposed principles. Regardless of one's position on this question, the ideas are compelling and provide a set of unifying concepts for thinking about adaptation.

Internal Models and Prediction

Internal models are central to Holland's theory of adaptive systems. He posited that all adaptive systems create and use internal models to prosper in the face of "perpetual novelty."

Models can be tacit and learned over evolutionary time, as in the case of bacteria swimming up a chemical gradient, or explicit and learned over a single lifespan, as in the case of cognitive systems that incorporate experience into internal representations through learning. In Holland's view, the key activity of an adaptive agent involves building and refining these data-driven models of the environment.

In his second book, *Induction*,[14] Holland and his co-authors proposed inductive methods by which cognitive agents can construct—and profit from—internal models by combining environmental inputs and rewards with stored knowledge. In their framework, an internal model defines a set of equivalence classes over states of the world, together with a set of transition rules between the equivalence classes, all of which are learned over time based on environmental rewards (or punishments). The many-to-one mapping from states of the world to states of the model (the equivalence classes) is called a homomorphism. Models that form valid homomorphisms with the part of the world being modeled allow the system to make accurate predictions. In Holland's conception, the equivalence classes are initially very general, as with, say, "moving object" and "stationary object." Through experience and learning, these classes can be specialized into more useful and precise subclasses, as in, say, "insect" and "nest." Over time, the adaptive system builds up a default hierarchy of rules covering general cases and refinements for specific classes.

At the time of Holland's work, the idea of default hierarchies was prevalent in knowledge representation systems. Holland made two key contributions to this literature. The first was his emphasis on homomorphisms as a formal way to evaluate model validity, an idea that dates back to W. Ross Ashby's *An Introduction to Cybernetics*.[2] Holland's student Bernard Ziegler developed this idea into a formal theory of computer modeling and simulation.[30] Even today, these early homomorphic theories of modeling stand as the most elegant approach we know of to characterize how consistent a model is with its environment and how an intelligent agent, human or artificial, can update a model to better reflect reality.

Holland's second key contribution was describing a computational mechanism, the "learning classifier system,"[13,21] to illustrate how a cognitive system could iteratively build up a detailed and hierarchical model of its environment to enhance survival. The key learning elements of this method, the "bucket-brigade" algorithm, combined with a genetic algorithm, presaged many of the ideas in modern reinforcement learning, notably the temporal-difference methods introduced by Sutton and Barto.[28]

Holland's inspiration for classifier systems came from several different disciplines, including Hebbian learning, artificial intelligence, evolutionary biology, economics, psychology, and control theory. Knowledge representation in the form of a population of "if-then" rules was a natural choice due to its popularity in AI at the time, as well as Holland's early work on modeling Hebbian cell assemblies: "In Hebb's view, a cell assembly makes a simple statement: If such and such an event occurs, then I will fire for a while at a high rate."[29] The if-then rules, when activated, compete to post their results on a shared "message list," serving as the system's short-term memory, again inspired by Hebb's work, as well as by AI blackboard systems of the day. Unlike blackboard systems, however, new rules are generated automatically in a trial-and-error fashion, and can be selected and recombined by a genetic algorithm.

Successful rules are strengthened over time if their predictions lead to positive rewards from the environment (and weakened otherwise) through a credit-assignment method called the bucket-brigade algorithm, in which rules gaining rewards from the environment or from other rules, transfer some of their gains to earlier-firing "stage-setting" rules that set up the conditions for the eventual reward. Holland credited Arthur Samuel's pioneering work on machine learning applied to checkers[26] as a key inspiration for these ideas.

Holland was primarily interested in how the two learning mechanisms—discovery of new rules and apportioning credit to existing rules—could work together to create useful default hierarchies of rules. He emphasized that the competition inherent in the learning and action mechanisms would allow the system to adapt to a continually changing environment without losing what it had learned in the past.

Holland put it this way: "Competition among rules provides the system with a graceful way of handling perpetual novelty. When a system has strong rules that respond to a particular situation, it is the equivalent of saying that it has certain well-validated hypotheses ... New rules do not interfere with the system's action in well-practiced situations but wait gracefully in the wings as hypotheses about what to do under novel circumstances."[15]

Although Holland proposed classifier systems as an executable theory of inductive processes in cognition, other researchers took them further, developing applications to areas as diverse as poker playing,[27] control of gas pipeline transmission,[9] and modeling the stock market.[24] (See Booker et al.[4] for more on practical applications of classifier systems.) Today, other reinforcement learning methods are more popular for real-world decision and control problems, but classifier systems can perhaps be viewed as an early stage-setting method that foreshadowed these later approaches.

Holland is best known for his invention of genetic algorithms, a family of search and optimization methods inspired by biological evolution.

Exploratory Modeling

Given that Holland believed the ability to learn and manipulate internal models was essential for any adaptive system, it is no surprise that he viewed modeling as essential for scientific inquiry.

Today, we use computational models both for prediction—by analyzing data via statistical models—and for understanding how systems work—by probing the effects of hypothesized underlying mechanisms. This latter use of models for enhancing understanding was dear to Holland's heart. In his view, the key to science is to understand the mechanisms that cause a system to behave in a certain way, an aspiration that goes well beyond data-fitting methods, which typically focus on the aggregate behavior of a system.

For example, a purely statistical model that describes the boom-and-bust pattern of the stock market does not address the underlying mechanisms that lead to these cycles through the collective actions of myriad individual buy/sell decisions. In contrast, the genetic algorithms for which Holland is so famous are exploratory models of mechanism; they provide a simple computational framework in which to explore the dynamics of Darwinian evolution and whether the basic mechanisms of variation, differential reproduction, and heredity are sufficient to account for the richness of the natural world.

Holland's work focused on exploratory models—those that explore basic principles and mechanisms, even if they do not make specific or detailed predictions.[16] Such models can show generically how certain behaviors could be produced. Holland pioneered a style of modeling that has come to be known as "individual-based" or "agent-based," in which every component of a system is represented explicitly (such as every trader in a stock market system or every cell in an immune system model) and has a dynamic internal state. In such models, each agent has its own behavior rules it can modify over time, through learning. In order to capture the behavior of systems under spatial constraints, these models are often defined over spatial structures (such as networks or simple grids).

The exploratory models championed by Holland were not intended to provide detailed, domain-specific predictions. They were meant instead to explore general mechanisms of complex systems and thus provide insights that might lead to more specific, detailed models. Such idealized models are akin to what philosopher Daniel Dennett has called "intuition pumps."[6]

The emphasis on exploratory models to build intuitions was an important theme of Holland's work, and he often quoted Sir Arthur Eddington's remark on the occasion of the first experimental test of Albert Einstein's theory of relativity: "The contemplation in natural science of a wider domain than the actual leads to a far better understanding of the actual."[7] This view of modeling is not typical. For many scientists, models are useful only to the extent they generate accurate predictions, a perspective that rules out the very kind of exploratory modeling that interested Holland the most.

Universal Properties of Complex Adaptive Systems

Holland was interested in a broad array of adaptive systems, including immune systems, ecologies, financial

markets, cities, and the brain. In the early 1980s, he teamed up with a small group of scientists, primarily physicists, with a sprinkling of economists and biologists, to discuss what properties this wide swath of "complex adaptive systems" have in common. The discussions helped define the intellectual mission of the Santa Fe Institute, the first institution dedicated to developing a science of complexity, as well as the other complexity institutes that followed. Holland brought to these discussions his lifelong study of adaptation and a reminder that serious theories about complexity would need to look deeper than phenomenological descriptions but also account for the "how" and "why" of these systems.

As the discussions about complex adaptive systems matured, a consensus developed about their basic properties. Such systems are composed of many components with nonlinear interactions; are characterized by complex emergent behavior; exhibit higher-order patterns; operate at multiple (and often nested) spatial and temporal scales, with some behavior conserved across all scales and other behaviors changing at different scales; and are adaptive, with behavioral rules continually adjusted through evolution and learning. Although this list is far from a formal characterization of complex adaptive systems, most people working in the field today are interested in systems that have these properties.

In the early 1990s, Holland teamed up with other Santa Fe Institute researchers, including several economists, to tackle the mismatch between the predictions of rational expectations theory—the dominant theory in economics at the time—and empirically observed stock-market behaviors. In brief, most economic theory of the day assumed that all participants in an economy or financial market are fully rational, acting to maximize their individual gain. In real life, however, actors in economies and markets are rarely wholly rational, and financial markets often deviate from rationality, as in, say, speculative bubbles and crashes.

The Santa Fe Institute Artificial Stock Market project[1,24] developed an exploratory model in which rational traders were replaced by adaptive traders—those who learn to forecast stock prices over time. The model tested for the emergence of three different trading strategies: fundamental, technical, and uninformed. The simulated market with adaptive trading agents was run many times, and the dynamics of price and trading volumes were compared to observed patterns in real markets. Holland and his collaborators found that the model's dynamics replicated several otherwise puzzling features of real-life markets.

Holland's view was that a system in stable equilibrium is essentially dead.

Although the Santa Fe Institute Stock Market model was highly simplified, it was very influential and led to many follow-on projects. It demonstrated clearly the essential role that adaptation plays in complex systems and illustrated how Holland's theories of continual learning in response to intermittent feedback from the environment could be integrated into domain-specific settings.

Echo[17,22] was an even more ambitious exploratory model Holland and his collaborators developed during the 1990s. Echo formalized Holland's idealization of complex adaptive systems into a runnable computational system where agents evolved external markers (called tags) and internal preferences, then used them to acquire resources and form higher-level aggregate structures (such as trading relationships, symbiotic groups, trophic cascades, and interdependent organizations). Echo agents sometimes discovered mimicry and used it to deceive competitors. These complex interacting patterns arose throughout the model's execution, which started with a single minimal agent. When the runs stabilized, the resulting population of agents was found to reproduce several well-known patterns observed in nature, most famously the rank-frequency distribution of species diversity known as the Preston curve in ecology. In lay terms, Echo evolved populations where "most species are rare."

The broad scope of the model, together with its ability to produce easily identifiable and well-known patterns from nature, was appealing to immunologists, economists, and evolutionary biologists alike. Many of the insights behind the project are described in Holland's third book, *Hidden Order*.[16]

Holland's later books, *Emergence*,[18] *Signals and Boundaries*,[19] and *Complex-*

ity: A Very Short Introduction[20] show how the theories of adaptation Holland developed during the earlier part of his career fit into the larger landscape of complex systems research. In these works, he especially emphasized the open-ended nature of complex systems, where change and adaptation are continual, systems co-evolve with each other, and ecological niches arise and decay spontaneously depending on resource availability and competition.

Holland's focus on understanding the mechanisms by which complex patterns emerge and change, rather than simply characterizing the patterns themselves (such as describing chaotic attractors or power laws), reflected his determination to get to the heart of complex adaptive systems. This determination represents the best of science. Holland's willingness to tackle the most difficult questions, develop his own formalisms, and use mathematics and simulation to provide insight sets a high bar for scientists in all disciplines.

Conclusion

John Holland was unusual in his ability to absorb the essence of other disciplines, articulate grand overarching principles, and then back them up with computational mechanisms and mathematics. Unlike most researchers, Holland moved seamlessly among these three modes of thinking, developing models that were years ahead of their time. A close reading of his work reveals the antecedents of many ideas prevalent in machine learning today (such as reinforcement learning in non-Markovian environments and active learning). His seminal genetic algorithm spawned the field of evolutionary computation, and his insights and wisdom helped define what are today referred to as the "sciences of complexity."

Holland's many books and papers have influenced scientists around the world and across many disciplines. Closer to home, he introduced several generations of students at the University of Michigan to computation in natural systems, an idea that even today remains somewhat controversial, despite successes in genetic algorithms for engineering design, biomimicry for robotics, abstractions of pheromone trails in ant colonies for optimization methods, and mechanisms from immunology to improve computer security.

Behind the ideas is the man himself. Everyone who knew John personally remembers the gleam in his eye when encountering a new idea; his willingness to talk to anyone, no matter how famous or obscure; and his incredible generosity and patience. His personality and humanity were somehow inextricably entangled with his intellectual contributions. Since his death in 2015, many of Holland's former students and colleagues have movingly described their desire to emulate his personal qualities as much as his scientific excellence. His ideas, intellectual passion, and personal approach will serve as beacons for research in intelligent and complex systems for many years to come.

Acknowledgments

We thank R. Axelrod, L. Booker, R. Riolo, K. Winter, and the anonymous reviewers for their careful reading of the manuscript and many helpful suggestions. Stephanie Forrest gratefully acknowledges the partial support of NSF (NeTS 1518878, CNS 1444500), DARPA (FA8750-15-C-0118), Air Force Office of Scientific Research (FA8750-15-2-0075), and the Santa Fe Institute. Melanie Mitchell gratefully acknowledges the partial support of NSF (IIS-1423651). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

1. Arthur, W.B. Holland, J.H., LeBaron, B.D., Palmer, R.G., and Tayler, P. Asset pricing under endogenous expectations in an artificial stock market. In *The Economy As an Evolving Complex System II*, W.B. Arthur, S. Durlauf, and D. Lane, Eds. Addison-Wesley, Reading, MA, 1997.
2. Ashby, W.R. *An Introduction to Cybernetics*. Chapman & Hall, London, 1956.
3. Bellman, R.E. *Adaptive Control Processes: A Guided Tour*. Princeton University Press, Princeton, NJ, 1961.
4. Booker, L.B., Goldberg, D.E., and Holland, J.H. Classifier systems and genetic algorithms. *Artificial Intelligence 40*, 1–3 (1989), 235–282.
5. Christiansen, F.B. and Feldman, M.W. Algorithms, genetics, and populations: The schemata theorem revisited. *Complexity 3*, 3 (1998), 57–64.
6. Dennett, D.C. Elbow Room: The Varieties of Free Will Worth Wanting. MIT Press, Cambridge, MA, 1984.
7. Eddington, A. *The Nature of the Physical World: Gifford Lectures, 1927*. Cambridge University Press, Cambridge, U.K., 1927.
8. Fisher, R.A. *The Genetical Theory of Natural Selection: A Complete Variorum Edition*. Oxford University Press, Oxford, U.K.,1930.
9. Goldberg, D.E. *Computer-Aided Gas Pipeline Operation Using Genetic Algorithms and Rule Learning*. Ph.D. Dissertation, University of Michigan, Ann Arbor, MI, 1983; http://www.worldcat.org/title/computer-aided-gas-pipeline-operation-using-genetic-algorithms-and-rule-learning/oclc/70390568
10. Holland, J.H. Outline for a logical theory of adaptive systems. *Journal of the ACM 9*, 3 (1962), 297–314.
11. Holland, J.H. Genetic algorithms and the optimal allocation of trials. *SIAM Journal on Computing 2*, 2 (1973), 88–105.
12. Holland, J.H. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. University of Michigan Press, Ann Arbor, MI, 1975.
13. Holland, J.H. Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In *Machine Learning: An Artificial Intelligence Approach, Volume 2*, R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, Eds. Morgan-Kaufman, San Francisco, CA, 1986, 593–623.
14. Holland, J.H. *Induction: Processes of Inference, Learning, and Discovery*. MIT Press, Cambridge, MA, 1989.
15. Holland, J.H. Genetic algorithms. *Scientific American 267* (July 1992), 44–50.
16. Holland, J.H. *Hidden Order: How Adaptation Builds Complexity*. Perseus Books, New York, 1995.
17. Holland, J.H. Echoing emergence: Objectives, rough definitions, and speculations for Echo-class models. In *Complexity: Metaphor, Models, and Reality*, G.A. Cowen, D. Pines, and D. Meltzer, Eds. Perseus Books, New York, 1999, 309–342.
18. Holland, J.H. *Emergence: From Chaos to Order*. Oxford University Press, Oxford, U.K., 2000.
19. Holland, J.H. *Signals and Boundaries: Building Blocks for Complex Adaptive Systems*. MIT Press, Cambridge, MA, 2012.
20. Holland, J.H. *Complexity: A Very Short Introduction*. Oxford University Press, Oxford, U.K., 2014.
21. Holland, J.H. and Reitman, J.S. Cognitive systems based on adaptive algorithms. *ACM SIGART Bulletin 63* (June 1977), 49.
22. Hraber, P.T., Jones, T., and Forrest, S. The ecology of Echo. *Artificial Life 3*, 3 (1997), 165–190.
23. London, R.L. *Who Earned First Computer Science Ph.D.?* blog@cacm (Jan. 15, 2013); http://cacm.acm.org/blogs/blog-cacm/159591-who-earned-first-computer-science-phd/fulltext
24. Palmer, R.G., Arthur, W.B., Holland, J.H., LeBaron, B., and Tayler, P. Artificial economic life: A simple model of a stock market. *Physica D: Nonlinear Phenomena 75*, 1–3 (1994), 264–274.
25. Rochester, N., Holland, J.H., Haibt, L.H., and Duda, W. Tests on a cell assembly theory of the action of the brain, using a large digital computer. *IRE Transactions on Information Theory 2*, 3 (1956), 80–93.
26. Samuel, A.L. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development 3*, 3 (1959), 210–229.
27. Smith, S.F. *A Learning System Based on Genetic Adaptive Algorithms*. Ph.D. Dissertation, University of Pittsburgh, Pittsburgh, PA, 1980.
28. Sutton, A.G. and Barto, R.S. Time derivative models of Pavlovian reinforcement. In *Learning and Computational Neuroscience: Foundations of Adaptive Networks*, M. Gabriel and J. Moore, Eds. MIT Press, Cambridge, MA, 1990, 497–537.
29. Waldrop, M.M. *Complexity: The Emerging Science at the Edge of Order and Chaos*. Simon and Schuster, New York, 1993.
30. Ziegler, B. *Theory of Modeling and Simulation*. Wiley Interscience, New York, 1976.

Stephanie Forrest (forrest@cs.unm.edu) is Distinguished Professor of Computer Science at the University of New Mexico, Albuquerque, NM, External Professor at the Santa Fe Institute, Santa Fe, NM, and a Fellow of the IEEE; John Holland was her Ph.D. advisor at the University of Michigan.

Melanie Mitchell (mm@pdx.edu) is Professor of Computer Science at Portland State University and External Professor at the Santa Fe Institute, Santa Fe, NM; John Holland was co-advisor for her Ph.D. at the University of Michigan.

DOI:10.1145/2888391

The skills and knowledge that earn promotions are not always enough to ensure success in the new position.

LEON KAPPELMAN, MARY C. JONES, VESS JOHNSON, EPHRAIM R. MCLEAN, AND KITTIPONG BOONME

Skills for Success

at Different Stages of an IT Professional's Career

SKILLED INFORMATION TECHNOLOGY (IT) professionals are scarce and costly. IT managers must not only ensure they hire the right entry-level capabilities and skills but also understand how to acquire, nurture, retain, and develop an evolving and diverse set of employee skills throughout the IT hierarchy. They must also ensure they develop their own skills to achieve the career progression they desire.[14] Much has been written about the skills needed for success in the IT field. The appropriate set of skills for entry-level new hires is of great interest to scholars, as well as to employers.[1,2,4,7,15,24,25] However, less attention has gone toward identifying skills required for the success of mid-level IT managers and CIOs beyond calls for excellent communication skills, strong business acumen, and strategic thinking.[6–9] Although these qualities are important, a more comprehensive set of skills is increasingly important, as organizations grapple with the costs of hiring, training, retention, and turnover.[5,7,12,13,18–20]

Though many studies have considered IT skills, most focus on specific technical skills (such as programming languages, hardware, and systems analysis) and are framed from the perspective of employers, IT degree programs, or recent IT graduates,[18,23] focusing primarily on skills for new hires.[1,2] Limited research has informed our understanding of the skills needed for an IT professional who wants to progress up the ranks of an organization. There is also little or no information on the expectations of senior and mid-level IT managers with regard to their own skills or of those above and below them in their chains of command. Research has found, however, that understanding what abilities are needed to progress successfully in one's career is critical to motivating IT employees.[10,22] Management and leadership skills gain increasing importance as one moves up the IT hierarchy. However, achieving the right balance of technical and managerial skills is difficult, and many organizations fail to provide employees adequate guidance toward achieving this critical skill mix.[3,10,11,16,22]

To address these shortcomings, the Society for Information Management (SIM) annual *IT Trends Study* included a number of questions in its 2014 membership questionnaire to provide a snapshot of the relative importance of a broad set of skills across three

» key insights

- **The key to progression in an IT management career is to hone one's technical and functional-area skills early.**
- **IT professionals must build their people and decision-making skills as they progress.**
- **Communication skills are critical throughout an IT professional's career.**

IMAGE BY QUINKY

levels, or stages, of IT careers: CIO or senior-most IT manager, mid-level IT professional, and newly hired IT employee. In particular, it captured the perspectives of CIOs and mid-level IT managers regarding what skills are most important for success at each of the stages. This research continues to provide insight into the skills IT professionals require to not only succeed in their current positions but move into higher levels of responsibility, and succeed there, too.

Expanding IT Skills

The task of identifying the skills needed by IT employees as they rise through the ranks of an organization is difficult because these skills are evolving continuously and can be job- or organization-specific.[8,15] The balance among technical, managerial, leadership, and business skills shifts in response to changing business and technology conditions.[3] In the 1960s and 1970s, driven by the predominance of mainframe com-

puters and long system development cycles, technical skills were the most valued, followed closely by managerial skills.[4,16] In the 1980s and 1990s, the widespread use of personal computers and local-area networks triggered a shift to a more strategic focus.[21] However, IT was still viewed primarily as a support activity responsible for managing the infrastructure needed for the functional groups within the organization.[3,11,15] This brought greater need for broader technical skills, along with greater awareness of business and technology changes, as well as greater strategic focus. In the 2000s, the emergence of social networking, mobile computing, and e-commerce made it even more important for IT managers to address organization issues.[12,17,25] Interpersonal, technical, and organizational skills have long been important for the success of IT professionals at all stages of their careers, but the relative mix of skills at each stage is not always clear or straightforward.[5,7–9,15,16,25] For example, although nontechnical skills are often viewed as more important for senior-management career success, technical skills are still highly valued, particularly in organizations that offer dual career ladders, allowing employees to pursue either technical or managerial career paths.[10,16,24]

Table 1. Respondents' work tenures.

	IT	Current Organization	Current Position
< 5 years	8.9%	44.4%	65.4%
>=5 to <10 years	5.6%	21.0%	22.4%
>=10 to <15 years	11.0%	15.2%	6.7%
>=15 to <20 years	18.3%	9.5%	4.4%
>=20 years	56.2%	9.8%	1.1%

Figure 1. Annual revenue of respondents' organizations ($U.S.).

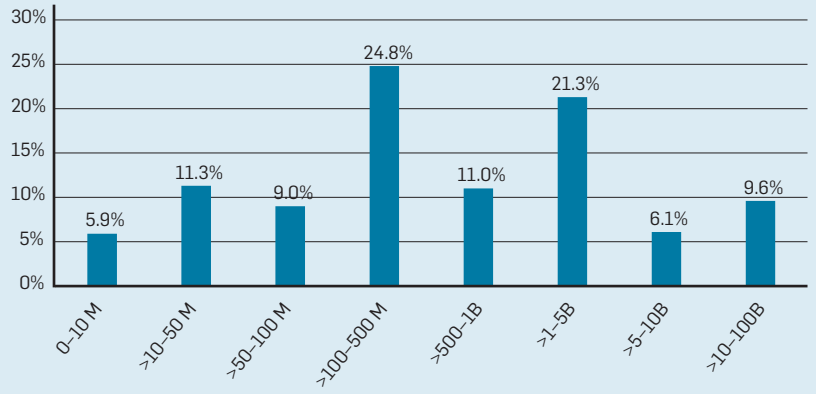
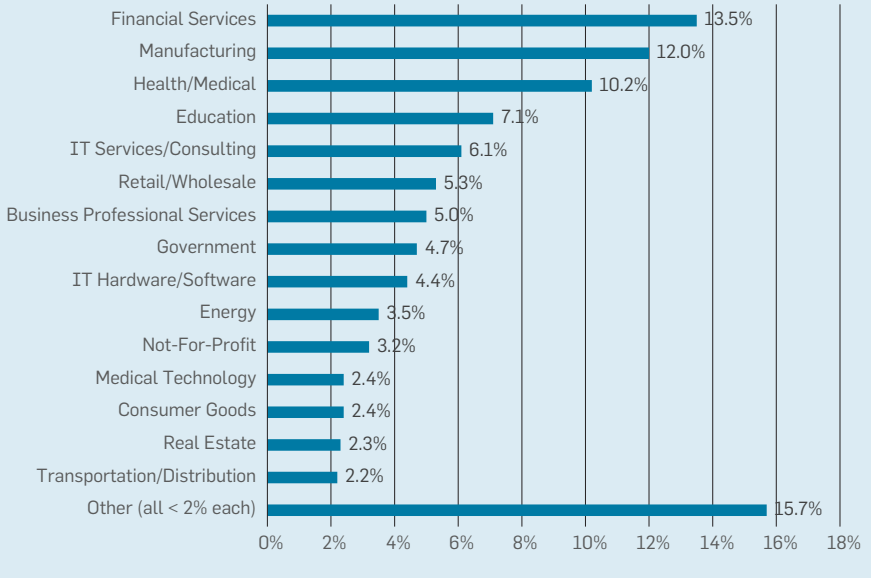


Figure 2. Industries represented by respondents.



Understanding this evolution of skill requirements is important for several reasons. First, as IT organizations become more strategic, understanding issues related to recruiting, developing, and retaining skilled IT employees becomes even more important.[5,8,13,15,21] Second, because IT has become more strategic and business-focused in most organizations, the skills needed at all levels of IT personnel have evolved and expanded. A good understanding of the skills needed at each career stage is thus important. IT employees must continue to acquire the technical skills needed to address advancements in technology but also acquire the business, management, and leadership skills necessary to move up.

Findings from the Field

We collected the data for this study as part of the larger *SIM IT Trends Study*.[12,13,17] We developed a questionnaire through a review of the literature, previous *SIM Trends* studies, and a multi-round Delphi method.[9,12,13,17,23,25] The Delphi panel, a characteristic of these SIM studies since their inception in the late 1970s, consisted of academic researchers and SIM-member IT practitioners. After each Delphi review, including pilot testing, the questionnaire was modified. The final online questionnaire was distributed by emailing a personal link to 4,612 SIM members in the spring of 2014. A total of 1,002 completed the questionnaire, a 21.4% response rate.

We received responses from 312 CIOs, 384 mid-level IT managers, and 306 practitioners who either did not answer the skills questions or self-identified as "others," or consultants, vendors, academics, retired, or in transition. "CIO" represented the highest-ranking IT person in an organization, regardless of title. The term "mid-level IT manager/professional" was used for respondents in IT management positions below CIO. Everyone in the sample was an IT manager since SIM is a professional society for IT managers. Table 1 reports the distribution of the respondents' tenure. For distribution by organizational revenue, see Figure 1, and by industry type, see Figure 2.

Participants were presented with a list of 36 skills and asked to identify the three most important for the success of new IT hires, mid-level IT managers, and themselves in their current positions. Table 2 reports the top 10 rankings and percentage of CIOs selecting each success skill for each career stage.

"Providing leadership" was selected by 34.3% of CIOs as one of the top three skills for success in their jobs and was highest ranked. Not surprisingly, the next most selected CIO skills were directly related to leadership and big-picture thinking: "people management," "strategic planning," and "decision making." Two of the skills CIOs perceived as most important to their jobs were also skills they perceived as highly important to both mid-level IT managers and new IT hires: "oral communication" and "collaboration."

CIOs indicated that many of the most important skills for mid-level IT professional success are also important for new IT hires; for example, "problem solving," "technical knowledge," and "functional area knowledge" were all top picks by CIOs for these levels, albeit with different rankings and selection frequencies. This suggests CIOs perceive the critical skills for new hires must be honed for them to move up the ranks. Surprisingly, managerial skills like "decision making," "providing leadership," and "planning" were not among those most frequently chosen by responding CIOs as critical for success at mid-level.

Mid-level IT managers also largely perceived the skills that are most critical to their own job success to be similar to the ones most important for the success of new IT hires, as in Table 3. However, they chose "collaboration with others," "technical knowledge," "problem solving," "oral communication," and "people management" less frequently for themselves than for new hires. Consistent with CIOs, they did not see leadership skills as particularly important for mid-level career success. A picture thus emerged of mid-level IT managers as having greater management responsibilities but not leadership responsibilities.

Table 4 reports the perceptions of mid-level IT managers who were either one, two, or three or more levels away from their CIOs. Perceptions of mid-level IT managers about the set of skills necessary for their job success were fairly consistent, regardless of their position within the mid-management ranks; however, the relative importance of each skill in terms of rank and percent selecting a skill were often quite different. All three levels indicated "collaboration with others," "functional area knowledge," "technical knowledge," and "problem solving" as the top-ranked skills. Managers who reported being closer to their CIOs seemed to attach a higher relative

Table 2. CIO perceptions of essential success skills at three stages of an IT career (n = 312).

CIO	% Selecting	IT Middle Management	% Selecting	New IT Hire	% Selecting
Providing leadership	34.29%	Collaboration with others	33.65%	Technical knowledge	47.44%
People management	29.49%	Problem solving	19.87%	Problem solving	39.42%
Strategic planning	23.72%	Technical knowledge	19.87%	Collaboration with others	38.46%
Decision making	23.40%	People management	17.63%	Functional area knowledge	22.12%
Communication (oral)	20.83%	Functional area knowledge	16.99%	Communication (oral)	18.27%
Collaboration with others	20.19%	Communication (oral)	16.99%	Honesty/ credibility	15.38%
Emotional intelligence	16.03%	Business analysis	16.67%	User relationship management	8.65%
Honesty/ credibility	15.38%	Project management	14.10%	Business analysis	8.65%
Business analysis	11.86%	Decision making	11.54%	Communication (written)	8.65%
Change management	11.22%	Honesty/ credibility	11.22%	Emotional intelligence	7.69%

Table 3. Mid-level IT manager perceptions of skills* (non-CIOs; n = 384).*

IT Middle Management Skills		New IT Hire Skills	
Skills	**% Selecting**	**Skills**	**% Selecting**
Collaboration with others	34.4%	Collaboration with others	41.7%
Functional area knowledge	22.1%	Technical knowledge	38.5%
Problem solving	20.6%	Problem solving	32.0%
People management	20.1%	Communication (oral)	22.1%
Technical knowledge	17.7%	Functional area knowledge	18.0%
Communication (oral)	16.9%	Honesty	17.2%
Emotional intelligence	14.3%	Business analysis	12.2%
Communication (written)	12.5%	Emotional intelligence	10.7%
Honesty	10.2%	Communication (written)	10.7%
Managing expectations	9.9%	Programming	9.4%

* Only the top 10 skills are included.

Figure 3. Perception of top skills for mid-level IT managers by level of manager.

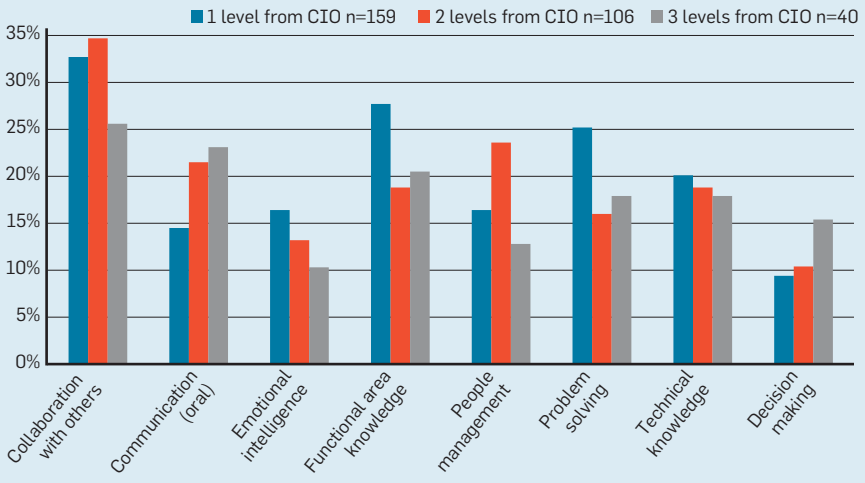


Table 4. Mid-level IS manager perceptions of own skills* by management level.**

1 level from CIO (n = 159)	% Selecting	2 levels from CIO (n = 106)	% Selecting	3 or more levels from CIO (n = 40)	% Selecting
Collaboration with others	32.7%	Collaboration with others	34.7%	Collaboration with others	25.6%
Functional area knowledge	27.7%	People management	23.6%	Communication (oral)	23.1%
Problem solving	25.2%	Communication (oral)	21.5%	Functional area knowledge	20.5%
Technical knowledge	20.1%	Functional area knowledge	18.8%	Problem solving	17.9%
Emotional intelligence	16.4%	Technical knowledge	18.8%	Technical knowledge	17.9%
People management	16.4%	Problem solving	16.0%	Decision making	15.4%
Communication (oral)	14.5%	Communication (written)	13.9%	Communication (written)	12.8%
Business analysis	10.7%	Emotional intelligence	13.2%	Honesty/credibility	12.8%
Honesty/credibility	10.7%	Marketing/sales	12.5%	Marketing/sales	12.8%
Communication (written)	10.1%	Project integration/ program management	11.8%	People management	12.8%

* Only the top 10 skills are included.
** 79 respondents did not provide number of levels away from CIO.

importance, as indicated by the higher percentage of respondents selecting an item, to "functional area knowledge" and "problem solving" than those at lower levels. Furthermore, one-level-away managers perceived "emotional intelligence" as one of the more important skills for their success, whereas managers three or more levels away perceived it as much less important. Conversely, the former attached less importance to both oral and written "communication" skills than those at lower levels. Lower-level managers reported "decision making" was a top skill, whereas far fewer of the higher-level managers rated it of critical importance to their personal success. Managers at a level farthest from their CIOs did not perceive "people management" as being as critical as the two higher levels did.

Figure 3 outlines the most important personal success skills at different reporting distances from the CIO, using the same data as Table 4 but including only those skills selected by at least 15% of the respondents. The eight skills in Figure 3 are also in the top six of at least one level in Table 4. Note how the relative importance of "emotional intelligence," "problem solving," and "functional area knowledge" increases as one moves up the IT hierarchy, while the importance of "oral communications" and "decision making" decreases.

It appears that as IT managers move up the career ladder and occupy positions closer to their CIOs, the set of skills they perceive as critical to their personal success remains fairly stable, even as the relative importance of individual skills can shift significantly. One possible explanation for this apparent contradiction is that as IT professionals gain more experience with a skill, they become less concerned about it, even though at each career stage, they are more focused on acquiring the skills they may not have needed in their prior positions (such as "functional area knowledge," "problem solving," and "emotional intelligence"). For example, "functional area knowledge," as selected by 27.7% of the respondents one level from the CIO, was their second most selected success skill, whereas lower-level managers did not rank it nearly as high. Third-ranked "problem solving" exhibited similar differences between those one level removed and the others. Higher-level IT managers may be more accountable to the business and thus must understand it better and more effectively solve its problems.

Limitations of the Study

One limitation is the data was self-reports of respondents' personal perceptions. Data could thus have been affected by respondents' experience and bias, including the industries and organization cultures they served. Respondents reported what they believe contributed to their success, but we have no independent evidence that these skills are the reason for that success. The generalizability of these

findings may be limited because all respondents were members of one professional association, SIM. Because SIM is an organization for IT professionals who have chosen a managerial career path, rather than a technical career path, findings might not be entirely applicable to those on a more technical track. On the other hand, a plus for generalizability is the significant degree of variation in the organization size and industries represented in the sample. However, since smaller organizations tend to have flatter hierarchies, size-related effects could be an issue for the mid-level perceptions in Table 3, Table 4, and Figure 3. Moreover, because we gathered only frequencies of responses, it was difficult to know with confidence the relative importance among the skills respondents selected; for example, if 20% of the respondents chose skill X and 10% chose skill Z, this does not necessarily mean skill X is twice as important as skill Z.

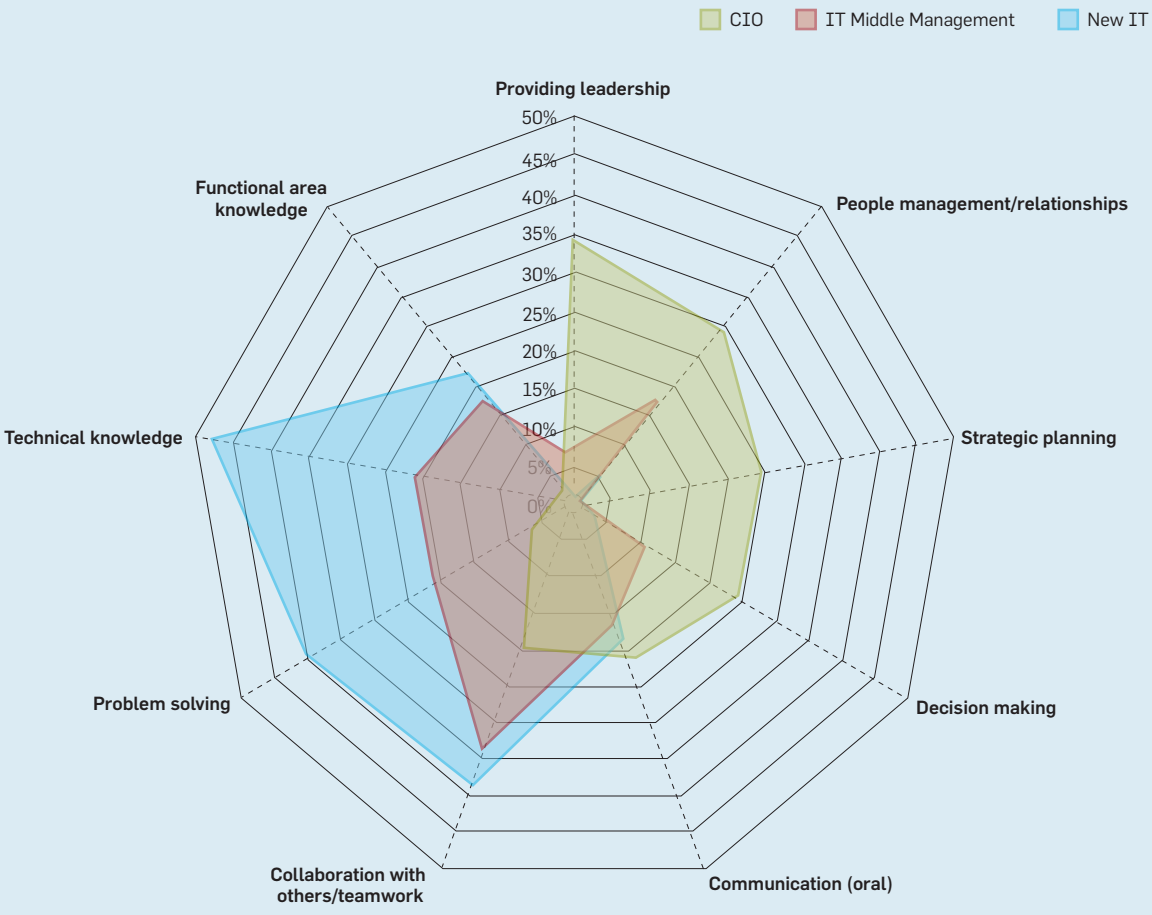
Implications and Lessons Learned

A picture emerges of the variety of skills IT managers deem necessary for success at various stages of an IT career, providing a broader, more dynamic view of skills for IT career success than the typical in-depth look at specific skills required for a given career stage or the skill shortages at a particular moment in time. The study also helps improve our understanding of the progression of skills needed as an IT professional's career advances from individual performer to manager of individual performers to leader.

This progression is reflected in Figure 4, illustrating the evolution of IT professional skill requirements as a career progresses and using the top five success skills most frequently selected by CIOs across the three career stages in Table 2. Figure 4 depicts how CIOs' perceptions of essential skills change for those IT professionals at different career stages. For example, "technical knowledge," "problem solving," and "collaboration" were the top skills for new IT hires yet were relatively less important for mid-level managers even though still highly ranked and significantly less important for CIOs, while "people management," "strategic planning," and "decision making" were of little importance to the success of new hires, somewhat important for mid-level managers, but were among the most important skills reported by respondents for CIO success. There is thus advance and decline in the relative importance of various skills as an IT professional's career progresses. However, Figure 4 also makes clear that throughout one's entire career, "collaboration" and "oral communications" skills remain important.

These findings, and recognition of the different skills needed at different career stages, represent an important contribution to our understand-

Figure 4. CIO perceptions of the top five skills across three levels of IT careers.



ing of IT skills and careers; they also have implications for organizations, as well as individuals. On the one hand, such understanding can help organizations improve their IT hiring practices and invest in designing and implementing appropriate training, education, retention, and career-advancement programs. On the other, they provide a framework whereby IT employees can develop and advance their own careers, acquiring the skills needed for success in their current positions and prepare for their next career stage.

In the first stage, new hires are expected to have the technical knowledge and problem-solving skills needed to perform the job. These skills would come from their college preparation in computer science or information systems, technical certifications (such as Cisco and Microsoft), or previous employment, internships, and work-study programs. At this stage, IT employees function as individual performers or members of a team. They also begin to gain the functional knowledge and business understanding needed to be a more effective performer.

At mid-level, IT professionals leverage their technical experience to be managers of individual performers, heading teams, IT projects, or programs. Here "decision making," "business knowledge," and "people management" skills become more important than they were previously. Obtaining an MBA or a master's degree in IS, or participating in personal- or management-development programs can be valuable career enhancers at this stage, enabling opportunities for greater management responsibilities.

Upon achieving executive leadership responsibilities, as in, say, a promotion to CIO, "providing leadership," along with "strategic planning" and "decision making," become key ingredients for success. It is here that knowledge of the business must be linked to knowledge of the business's customers, suppliers, and indeed the industry itself. Here, senior management and executive leadership programs (such as at Harvard, MIT, Stanford, and SIM's Regional Leadership Forum) can help provide this broader perspective.

All this highlights the importance of continuous, lifelong learning to the success of IT professionals. In addition to the formal academic programs discussed here, mentoring by a more senior manager in the organization (but not necessarily one's boss) can be very helpful toward career advancement. If not formally provided by an employer, individuals should seek out such mentors informally through professional associations like ACM and personal networks. Similarly, external career coaches can be helpful, particularly at critical decision points in one's career.

Conclusion

This research study, which was based on SIM member data, provides insights into the diverse and dynamic nature of skill requirements at different stages of an IT career and levels of responsibility from the perspective of most senior and mid-level IT managers. Organizations can use them to enhance their IT workforce practices and IT professionals to achieve their personal career objectives and help others do so, too. C

References

1. Aasheim, C.L., Li, L., and Williams, S. Knowledge and skill requirements for entry-level information technology workers: A comparison of industry and academia. *Journal of Information Systems Education 20*, 3 (Summer 2009), 349–356.
2. Aasheim, C.L., Shropshire, J., Li, L., and Kadlee, C. Knowledge and skill requirements for entry-level IT workers: A longitudinal study. *Journal of Information Systems Education 23*, 2 (Spring 2012), 193–204.
3. Agarwal, R. and Ferratt, T. Enduring practices for managing IT professionals. *Commun. ACM 45*, 9 (Sept. 2002), 73–79.
4. Benamati, J.H., Ozdemir, Z.D., and Smith, H.J. Aligning undergraduate IS curricula with industry needs. *Commun. ACM 53*, 3 (Mar. 2010), 152–156.
5. Byrd, T.A. and Turner, D.E. An exploratory analysis of the value of the skills of IT personnel: Their relationship to IS infrastructure and competitive advantage. *Decision Sciences 32*, 1 (Mar. 2001), 21–54.
6. Clark, C.E., Cavanaugh, N.C., Brown, C.V., and Sambamurthy, V. Building change-readiness capabilities in the IS organization: Insights from the Bell Atlantic experience. *MIS Quarterly 21*, 4 (Dec. 1997), 425–454.
7. Gallagher, K.P., Kaiser, K.M., Simon, J.C., Beath, C.M., and Goles, T. The requisite variety of skills for IT professionals. *Commun. ACM 53*, 6 (Virtual Extension, June 2010), 144–148.
8. Harris, J. Preparing to be the chief information officer. *Journal of Leadership, Accountability and Ethics 8*, 5 (Dec. 2011), 56–62.
9. Hawk, S., Kaiser, K.M., Goles, T., Bullen, C.V., Simon, J.C., Beath, C.M., Gallagher, K.P., and Frampton, K. The information technology workforce: A comparison of critical skills of clients and service providers. *Information Systems Management 29*, 1 (2012), 2–12.
10. Hsu, M.K., Jiang, J.J., Klein, G., and Tang, Z. Perceived career incentives and intent to leave. *Information and Management 40*, 5 (May 2003), 361–369.
11. Igbaria, M., Greenhaus, J.H., and Parasuraman, S. Career orientations of MIS employees: An empirical analysis. *MIS Quarterly 15*, 2 (June 1991), 151–169.
12. Kappelman, L., McLean, E., Johnson, V., and Gerhart, N. The 2014 SIM IT Key Issues and Trends Study. *MIS Quarterly Executive 13*, 4 (Dec. 2014), 237–263.
13. Kappelman, L., McLean, E., Luftman, J., and Johnson, V. IT key issues, investments, and practices of organizations and IT executives: Results and observations from the 2013 SIM IT Trend Study. *MIS Quarterly Executive 12*, 4 (Dec. 2013), 227–240.
14. Lee, C.K. and Wingreen, S.C. Transferability of knowledge, skills, and abilities along IT career paths: An agency theory perspective. *Journal of Organizational Computing and Electronic Commerce 20*, 1 (Feb. 2010), 23–44.
15. Lee, D.M.S., Trauth, E.M., and Farwell, D. Critical skills and knowledge requirements of IS professionals: A joint academic/industry investigation. *MIS Quarterly (Special Issue on IS Curricula and Pedagogy) 19*, 3 (Sept. 1995), 313–340.
16. Litecky, C.R., Arnett, K.P., and Prabhakar, B. The paradox of soft skills versus technical skills in IS hiring. *Journal of Computer Information Systems 45*, 1 (Sept. 2004), 69–76.
17. Luftman, J., Kempaiah, R., and Nash, E. Key issues for IT executives 2005. *MIS Quarterly Executive 5*, 2 (June 2006), 27–45.
18. Overby, S. CIOs struggle with the great talent hunt. *CIO* (May 1, 2013), 32–44.
19. Prabhakar, B., Litecky, C.R., and Arnett, K. IT skills in a tough job market. *Commun. ACM 48*, 10 (Oct. 2005), 91–94.
20. Schwarzkopf, A.B., Mejias, R.J., Jasperson, J.S., Saunders, C.S., and Gruenwald, H. Effective practices for IT skills staffing. *Commun. ACM 47*, 1 (Jan. 2004), 83–88.
21. Sethi, V. and King, W.R. Development of measures to assess the extent to which an information technology application provides competitive advantage. *Management Science 40*, 12 (Dec. 1994), 1601–1627.
22. Smits, S.J., McLean E.R., and Tanner, J.R. Managing high-achieving information systems professionals. *Journal of Management Information Systems 9*, 4 (1993), 103–120.
23. Todd, P.A., McKeen, J.D., and Gallupe, R.B. The evolution of IS job skills: A content analysis of IS job advertisements from 1970 to 1990. *MIS Quarterly 19*, 1 (Mar. 1995), 1–27.
24. Wilkerson, J. An alumni assessment of MIS related job skill importance and skill gaps. *Journal of Information Systems Education 23*, 1 (2012), 85–99.
25. Zwieg, P., Kaiser, K.M., Beath, C.M., Bullen, C., Gallagher, K.P., Goles, T., Howland, J., Simon, J.C., Abbot, P. Abraham, T., Carmel, E., Evaristo, R. Hawk, S., Lacity, M., Gallivan, M., Kelly, S., Mooney, J.G., Ranganathan, C., Rottman, J.W., Ryan, T., and Wion, R. The information technology workforce: Trends and implications 2005–2008. *MIS Quarterly Executive 5*, 2 (June 2006), 47–54.

Leon Kappelman (Leon.Kappelman@unt.edu) is a professor in the Department of Information Technology & Decision Sciences in the College of Business at the University of North Texas, Denton, TX.

Mary C. Jones (mary.jones@unt.edu) is department chair and a professor in the Department of Information Technology & Decision Sciences in the College of Business at the University of North Texas, Denton, TX.

Vess Johnson (vljohnso@uiwtx.edu) is an assistant professor and MIS coordinator at the H-E-B School of Business at the University of the Incarnate Work, San Antonio, TX.

Ephraim R. McLean (emclean@gsu.edu) is a Regents' professor and George E. Smith Eminent Scholar's chair in Computer Information Systems Department in the J. Mack Robinson College of Business at Georgia State University, Atlanta, GA.

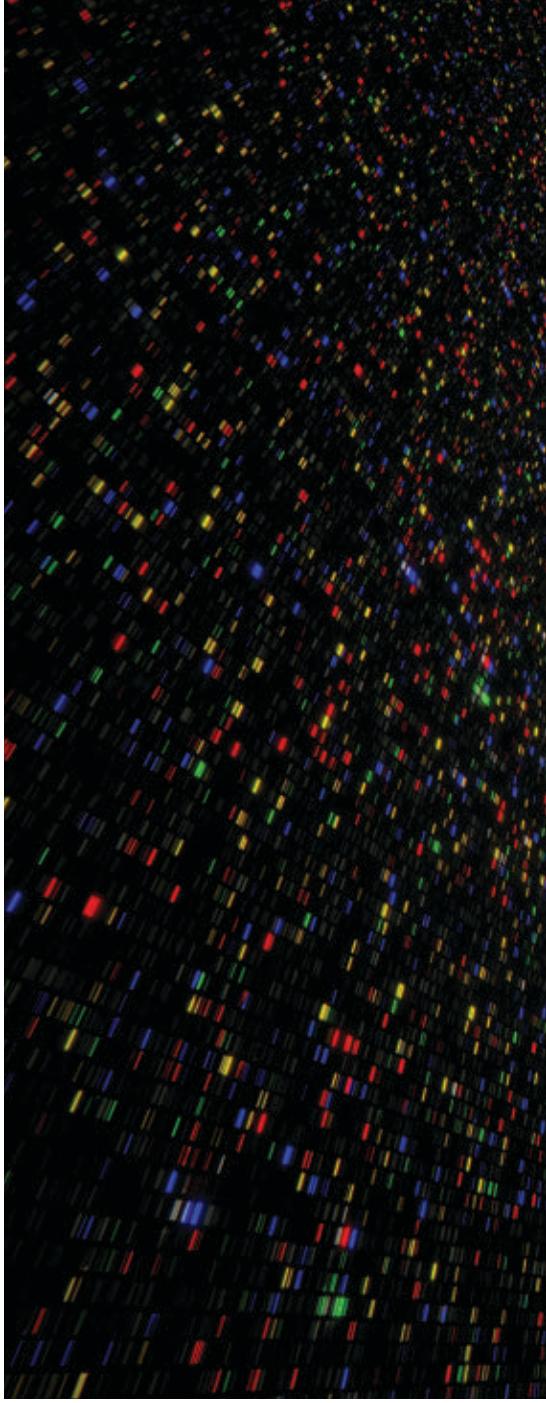
Kittipong Boonme (kboonme@twu.edu) is an assistant professor in the School of Management at Texas Woman's University, Denton TX.

DOI:10.1145/2957324

Algorithmic advances take advantage of the structure of massive biological data landscape.

BY BONNIE BERGER, NOAH M. DANIELS, AND Y. WILLIAM YU

Computational Biology in the 21st Century: Scaling with Compressive Algorithms



COMPUTATIONAL BIOLOGISTS ANSWER biological and biomedical questions by using computation in support of—or in place of—laboratory procedures, hoping to obtain more accurate answers at a greatly reduced cost. The past two decades have seen unprecedented technological progress with regard to generating biological data; next-generation sequencing, mass spectrometry, microarrays, cryo-electron microscopy, and other high-throughput approaches have led to an explosion of data. However, this explosion is a mixed blessing. On the one hand, the scale and scope of data should allow new insights into genetic and infectious diseases, cancer, basic biology, and even human migration patterns. On the other hand, researchers are generating datasets so massive that it has become

» key insights

- There is a lot of commonality in sequences and other biological data—even more redundancy than in a text file of the English language.
- This means we can take advantage of compression algorithms that exploit that commonality and represent many sequences by only a few bits.
- Of course, we are dealing with a massive amount of data so that compression becomes important for efficiency.
- We highlight recent research that capitalizes on structural properties of biological data—low metric entropy and fractal dimension—to allow us to design algorithms that run in sublinear time and space.

ILLUSTRATION BY PETER CROWTHER ASSOCIATES

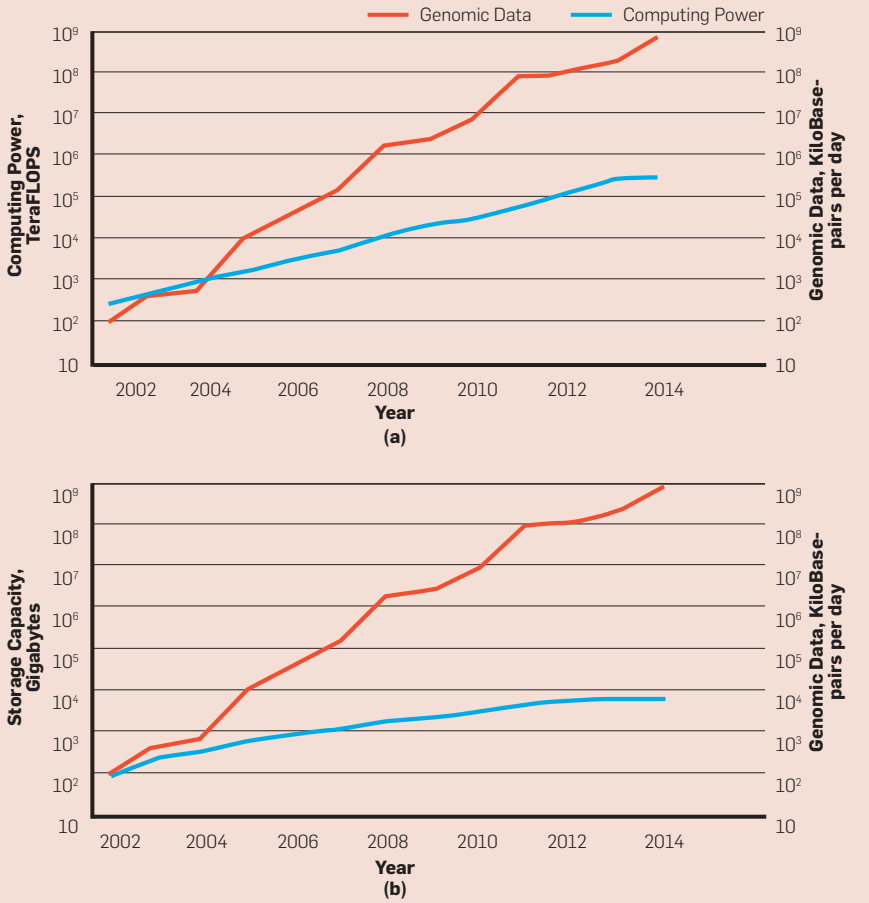
difficult to analyze them to discover patterns that give clues to the underlying biological processes.

Certainly, computers are getting faster and more economical; the amount of processing available per dollar of computer hardware is more or less doubling every year or two; a similar claim can be made about storage capacity (Figure 1).

In 2002, when the first human genome was sequenced, the growth in computing power was still matching the growth rate of genomic data. However, the sequencing technology used for the Human Genome Project—Sanger sequencing—was supplanted around 2004, with the advent of what is now known as next-generation sequencing. The material costs to sequence a genome have plummeted in the past decade, to the point where a whole human genome can be sequenced for less than US$1,000. As a result, the amount of genomic data available to researchers is increasing by a factor of 10 every year.

This growth in data poses significant challenges for researchers.[25] Currently, many biological "omics" applications require us to store, access, and analyze large libraries of data. One approach to solving these challenges is to embrace cloud computing. Google, Inc. and the Broad Institute have collaborated to bring the GATK (Genome Analysis Toolkit) to the Google cloud (https://cloud.google.com/genomics/gatk). Amazon Web Services are also commonly used for computational biology research and enterprise (for example, DNAnexus).[31] However, while cloud computing frees researchers from maintaining their own datacenters and provides cost-saving benefits when computing resources are not needed continuously, it is no panacea. First and foremost, the computer systems that make up those cloud datacenters are themselves bound by improvements in semiconductor technology and Moore's Law. Thus, cloud computing does not truly address the problem posed by the faster-than-Moore's-Law exponential growth

Figure 1. (a) Moore's and (b) Kryder's laws contrasted with genomic sequence data.



in omics data. Moreover, in the face of disease outbreaks such as the 2014 Ebola virus epidemic in West Africa, analysis resources are needed at often-remote field sites. While it is now possible to bring sequencing equipment and limited computing resources to remote sites, Internet connectivity is still highly constrained; accessing cloud resources for analytics may not be possible.

Computer scientists routinely exploit the structure of various data in order to reduce time or space complexity. In computational biology, this approach has implicitly served researchers well. Now-classical approaches such as principal component analysis (PCA) reduce the dimensionality of data in order to simplify analysis and uncover salient features.[3] As another example, clever indexing techniques such as the Burrows-Wheeler Transform (BWT) take advantage of aspects of sequence structure[3] to speed up computation and save storage. This article focuses on cutting-edge algorithmic advances for dealing with the growth in biological data by explicitly taking advantage of its unique structure; algorithms for gaining novel biological insights are not its focus.

Types of Biological Data

In the central dogma of molecular biology, DNA is transcribed into RNA, which is translated by the ribosome into polypeptide chains, sequences of amino acids, which singly or in complexes are known as proteins. Proteins fold into sophisticated, low-energy structures, which function as cellular machines; the DNA sequence determines the amino acid sequence, which in turn determines the folded structure of a protein. This structure ultimately determines a protein's function within the cell. Certain kinds of RNA also function as cellular machines. Methods have been developed to gather biological data from every level of this process, resulting in a massive influx of data on sequence, abundance, structure, function, and interaction of DNA, RNA, and proteins. Much of this data is amenable to standard Big Data analysis methods; however, in this article we focus on examples of biological data that exhibit additional exploitable structure for creating scalable algorithms.

Sequence data, either nucleotide sequences (using a four-letter alphabet representing the four DNA or RNA bases) or protein sequences (using a 20-letter alphabet representing the 20 standard amino acids) are obtained in several ways. For both protein and RNA sequence data, mass spectrometry, which can determine protein sequence and interactions and RNA-seq, which can determine RNA sequence and abundance allow scientists to also infer the expression of the gene to which it might translate play central roles. However, with the advent of next-generation sequencing (NGS) technologies, the greatest volume of sequence data available is that of DNA. To better understand the structure of NGS sequence data, we will expand on NGS methodologies.

At the dawn of the genomic era, Sanger sequencing was the most widely used method for reading a genome. More recently, however, NGS approaches, beginning with Illumina's "sequencing by synthesis," have enabled vastly greater throughput due to massive parallelism, low cost, and simple sample preparation. Illumina sequencing and other NGS approaches such as SOLiD, Ion Torrent, and 454 pyrosequencing do not read a single DNA molecule end-to-end as one could read through a bound book. Instead, in *shotgun sequencing*, DNA molecules are chopped into many small fragments; from these fragments we generate *reads* from one or both ends (Figure 2a). These reads must be put together in the correct order to piece together an entire genome. Current reads typically range from 50 to 200 bases long, though longer reads are available with some technologies (for example, PacBio). Because no sequencing technology is completely infallible, sequencing machines also provide a *quality score* (or measure of the confidence in the DNA base called) associated with each position. Thus, an NGS read is a string of DNA letters, coupled with a string of ASCII characters that encode the quality of the base call. A sequencing run will produce many overlapping reads.

While measuring abundance to generate gene expression data (for more information, see the Source Material that accompanies this article in the ACM Digital Library) lends itself to cluster analysis and probabilistic approaches, the high dimensionality and noise in the data present significant challenges. Principal Component Analysis has shown promise in reducing the dimensionality of gene expression data. Such data and its challenges have been the focus of other articles,[3] and thus will be only lightly touched upon here.

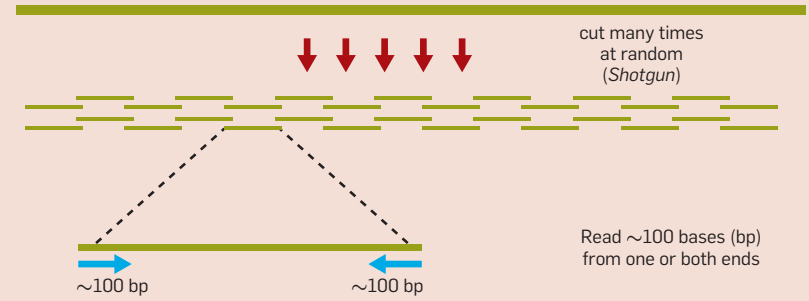
As mentioned earlier, function follows form, so in addition to sequence and expression, structure plays an important role in biological data science. However, we are not interested in only RNA and protein structures; small chemical compounds represent an additional source of relevant structural data, as they often interact with their larger RNA and protein brethren. Physical structures of molecules can be determined by X-ray crystallography, NMR, electron microscopy, and other techniques. Once determined, there are a variety of ways of representing these structures, from labeled graphs of molecular bonds to summaries of protein domains. These representations can then be stored in databases such as Pub-Chem or the Protein Data Bank, and are often searched through, for example, for potential small molecule agonists for protein targets. Importantly, as we will expand upon later, interesting biomolecules tend to be sparse and non-randomly distributed in many representational spaces, which can be used for accelerating the aforementioned searches.

When examining more complex phenomena than single proteins or compounds, we often look to synthesize things together into a systems-level understanding of biology. To that end, we frequently use networks to represent biological data, such as the genetic and physical interactions among proteins, as well as those in metabolic pathways.[3] While standard network science tools have been employed in these analyses—for example, several approaches make use of diffusion or random walks to explore the topology of networks[9,11]—they are often paired with more specific biological data, as seen in IsoRank[32] and IsoRankN's[21] use of conserved biological function in addition to random walks for global multiple network alignment. Other tools solve other biological problems, such as MONGOOSE,[10] which analyzes metabolic networks. However, given its breadth, biological network science is beyond the scope of this article.

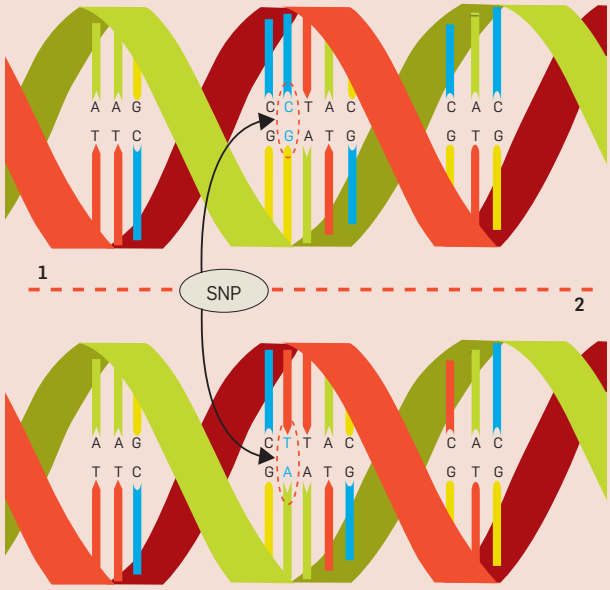
Challenges with Biological Data

Given DNA or RNA reads from NGS technologies, the first task is to *assemble* those fragments of sequence into contiguous sequences. The assembly problem is analogous to the problem of reconstructing a book with all its pages torn out. *De novo* assembly is beyond the scope of this article, but is possible because the sequence is covered by many overlapping reads;[3] for this task, the *de Bruijn graph* data structure is commonly used.[6] Often, however, a reference genome (or in the case of RNA, *transcriptome*) is available for the organism being sequenced; the

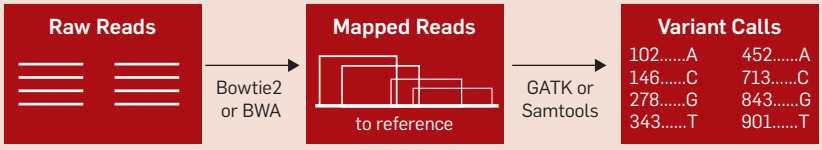
Figure 2. The next-generation sequencing (NGS) pipeline.



(a) 'Shotgun' sequencing breaks DNA molecules into many short fragments, which are read from one or both ends in the form of reads, and relies on high coverage to produce a statistically likely representation of a whole genome.



(b) Single-nucleotide polymorphisms, or SNPs, are the simplest type of genomic variant, and form the bulk of 'variant-calling' analysis.



(c) The NGS downstream analysis pipeline. Shotgun reads are mapped to a reference genome with tools such as BWA, Bowtie, or CORA. The resulting genomic sequence is analyzed for variants with tools such as GATK or Samtools. This allows relationships between genes and diseases to be uncovered.

establishment of a human reference genome was indeed the purpose of the Human Genome Project.

When a reference sequence is available, NGS reads can be mapped onto this reference (Figure 2c). Continuing the book analogy, it is much easier to reconstruct a book with all its pages torn out when one has another (perhaps imperfect) copy of that book to match pages to. Mapping allows the differences between the newly sequenced genome and the reference to be analyzed; these differences, or variants, may include single-nucleotide polymorphisms (SNPs, which are the genetic analogue to bit-flips, see Figure 2b), insertions or deletions, or larger-scale changes in the genome. Determining the differences between an individual genome and a reference is known as *variant calling*. While reference-based read mapping is a fundamentally simpler problem than de novo assembly, it is still computationally complex, as gigabytes or terabytes of reads must each be mapped onto the reference genome, which can range from millions (for bacteria) to billions (for mammals) of base pairs. As an example, the ICGC-TCGA Pan Cancer Analysis of Whole Genomes (PCAWG)[36] brings together more than 500 top cancer researchers from about 80 institutions in a coordinated manner with the goal of mapping the entire mutational landscape of 37 common cancer types. Currently, each sample requires seven hours to download even on an institutional connection. Importantly, researchers do not trust the provided mapping, and thus they redo mappings. The time spent on mapping is about 50% of the overall time spent on the sequence analysis pipeline. As read mapping is typically the most costly step in NGS analysis pipelines (for example, GATK), any improvement to existing mappers will immediately accelerate sequence analysis studies on large read datasets.

Driven by the plummeting costs of next-generation sequencing, the 1000 Genomes Project[1] is pursuing a broad catalog of human variation; instead of producing a single reference genome for a species, many complete genomes are catalogued. Likewise, WormBase and FlyBase are cataloguing many different species and strains of the *Caenorhabditis* worm and *Drosophila* fruit fly, respectively. These genomes are enabling cross-species inference, for example about genes and regulatory regions, and thus insights into function and evolution.[3] Again, the sheer enormity of sequencing data is problematic for storage, access, and analysis.

When examining more complex phenomena than single proteins or compounds, we often look to synthesize things together into a systems-level understanding of biology. To that end, we often use networks to represent biological data.

Given a sequenced genome, the next natural questions ask what genes (genomic regions that code for proteins) are present, what structure each resulting protein takes, and what biological function it performs. Identifying likely genes is a well-studied problem[3] beyond the scope of this article. However, determining evolutionary relationships, structure, and function is at the heart of current research in computational biology. Since some organisms (known as *model organisms*) are better studied than others, and evolution is known to conserve sequence, structure, and function, a powerful approach to determine these attributes is to search for similar sequences about which more is known. This so-called *homology search* entails searching for approximate matches in databases of known gene or protein sequences. The homology search problem was believed to be solved previously; Basic Local Alignment Search Tool (BLAST)[3] has been the standard tool for performing homology (similarity) search on databases of nucleotide and protein sequences. BLAST takes a "seed-and-extend" approach; it looks for small, *k*-mer matches that might lead to longer matches, and greedily extends them, ultimately producing a sequence alignment between a query and each potential database hit. However, BLAST's running time scales linearly with the size of the database being searched, which is problematic as sequence databases continue to grow at a faster rate than Moore's Law.

On a potentially even larger scale is the growth of *metagenomic* data. Metagenomics is the study of the many genomes (bacterial, fungal, and even viral) that make up a particular environment. Such an environment could be soil from a particular region (which can lead to the discovery of new antibiotics[14]), or it could be the human gut, whose microbiome has been linked to human-health concerns including Autism Spectrum Disorder,[23] Crohn's Disease, and obesity.

Metagenomics fundamentally asks what organisms are present, and, in the case of a microbiome such as the gut, what metabolic functions it can accomplish as a whole. One way of addressing this problem is to attempt to map NGS reads from a metagenomic sample onto a set of reference genomes that are expected to be present. This is exactly the read-mapping problem discussed early, but with many reference genomes, compounding the computational requirements. A second way is to perform homology search on a protein sequence database; exact or nearly exact matches imply the presence of a species, while more distant hits may still give clues to function. For this task, BLASTX[2] is commonly used to translate nucleotide reads into their possible protein sequences, and search for them in a protein database. The difficulty is the datasets required to shine any light on these questions, namely from "shotgun" metagenomics, are gigantic and vastly more complex than standard genomic datasets. The massive data results in major identification challenges for certain bacterial, as well as viral, species, and genera.[19]

The computational study of drugs and their targets based on chemical structure and function is known as *chemogenomics*.[5] In the fields of drug discovery and drug repurposing, the prediction of biologically active compounds is an important task. Computational high-throughput screening eliminates many compounds from laborious wet-lab consideration, but even computational screening can be time consuming.

Chemogenomics typically relies on comparing chemical graph structures to identify similar molecules and binding sites. Furthermore, comparing chemical graph structures typically involves computing the maximal common subgraph (MCS), an NP-hard problem. However, there are an increasing number of such chemical compounds to search; the NCBI's PubChem database has grown from 31 million compounds in January 2011 to 68 million in July 2015.

The continued ability to store, search, and analyze these growing datasets hinges on clever algorithms that take advantage of the structure of, and redundancy present in, the data. Indeed, these growing datasets "threaten to make the arising problems computationally infeasible."[3]

Definitions

Chemogenomics: **Computational study of drugs and their targets based on chemical structure and function.**

Metagenomics: **Study of the many genomes that make up a particular environment.**

Shotgun sequencing: **Modern genomic sequencing, which chops DNA into many short pieces**

Homology search: **Determining the function, structure, or identity of a gene sequence by locating similar sequences within an annotated database.**

Transcriptome: **Transcribed RNA from a genome, which results in protein production.**

BLAST: **Standard biological sequence similarity search tool.**

State-of-the-Art Approaches to Meet These Challenges

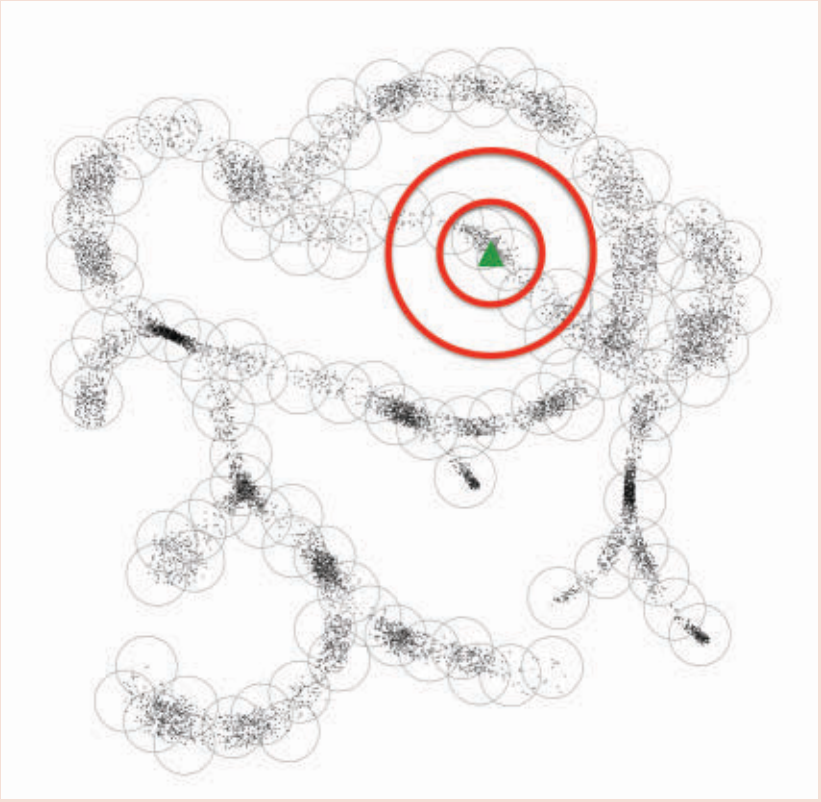
Techniques for reference-based read mapping typically rely on algorithmic approaches such as the Burrows-Wheeler transform (BWT), which provides efficient string compression through a reversible transformation, while the FM-index data structure is a compressed substring index, based on the BWT, which provides efficient storage as well as fast search.[3] BWA (Burrows-Wheeler Aligner) uses the BWT, while the Bowtie[2] mapper further relies on the FM-index for efficient mapping of NGS reads.[3] The Genome Multitool (GEM) mapper[24] also uses an FM-index coupled with dynamic programming in a compressed representation of the reference genome, in order to prune the search space when mapping reads to a reference genome. Masai[33] and mrsFAST[15] use an "approximate seed" approach to index the space of possible matches, likewise pruning the search space; however, the bulk of its runtime is spent on the extend phase. State-of-the-art mapper mrsFAST-Ultra achieves improvements in efficiency based on machine architecture rather than leveraging redundancy in the data itself with near-perfect sensitivity, but only for the case where there are no insertions and deletions (indels).[16] Even with these approaches, read mapping remains a significant bottleneck in genomic research.[3]

Compressing reads for storage is necessary should researchers wish to apply more advanced mapping tools or other analysis in the future.[4] As stated earlier, NGS reads consist of a sequence string and associated quality scores, the latter of which generally uses more space when compressed. By taking advantage of biological structure, both parts of NGS reads can be better compressed. Unlike some other approaches to compressing quality scores in the literature,[4,26] Quartz[39] takes advantage of the fact that midsize *l*-mers can in many cases almost uniquely identify locations in the genome, bounding the likelihood that a quality score is informative and allowing for lossy compression of uninformative scores. Because Quartz's lossy compression injects information from the distribution of *l*-mers in the target genome, it demonstrates not only improved compression over competing approaches, but slightly improves the accuracy of downstream variant-calling.[39] Similarly, for the sequence component of the read, Mince[27] takes advantage of sequence redundancy by grouping similar reads (those that share a common short—15bp—substring) together into buckets, allowing that common substring to be removed and treated as the bucket label, so that each read in the compressed representation comprises only its unique differences from the bucket label. This approach allows a general-purpose compressor to achieve better compression. SCALCE[3] also relies on a "boosting" scheme, reordering reads in such a way that a general-purpose compressor achieves improved compression.

Recent advances in metagenomic search tools have relied on two improvements over BLASTX: indexing and alphabet reduction. RapSearch2[40] relies on alphabet reduction and a collision-

Figure 3. Cartoon depiction of points in an arbitrary high-dimensional space, as might arise from genomes generated by mutation and selection during the course of evolution.

Although high dimensional locally, at the global scale of covering spheres, the data cloud looks nearly 1-dimensional, which enables entropy scaling of similarity search. Clusters cover the data points but do not cover unoccupied regions of space. The green triangle represents a query, with two concentric search radii (red circles) around it. Thanks to low fractal dimension, the large circle does not contain vastly more points than the small circle.



free hash table. The alphabet reduction, as it is reversible, can be thought of as a form of lossless compression; a 20-letter amino acid alphabet is mapped onto a smaller alphabet, with offsets stored to recover the original sequence in the full alphabet. The hash table provides an efficient index of the database to be searched. DIAMOND[7] also relies on alphabet reduction, but uses "shaped seeds"—essentially, *k*-mers of length 15–24 with wildcards at 9–12 specific positions—instead of simple *k*-mer seeds to index the database. DIAMOND demonstrates search performance three to four orders of magnitude faster than BLASTX, but still linear in the size of the database being searched.

Recent work on gene expression has explored additional ways to exploit the high-dimensional structure of the data. SPARCLE (SPArse ReCovery of Linear combinations of Expression)[28] brings ideas from compressed sensing[8] to gene expression analysis. Another recent and novel approach to exploiting the structure of gene expression space is Parti (Pareto task inference),[17] which describes a set of data as a polytope, and infers the specific tasks represented by vertices of that polytope from the features most highly enriched at those vertices.

The most widely used chemogenomics search is the Small Molecule Subgraph Detector (SMSD),[29] which applies one of several MCS algorithms based on the size and complexity of the graphs in question. Notably, large chemical compound databases, such as PubCHEM, cannot be searched on a laptop computer with current tools such as SMSD.

Structure of Biological Data

Fortunately, biological data has unique structure, which we later take advantage of to perform search that scales sublinearly in the size of the database.[38] The first critical observation is that much biological data is highly redundant; if a computation is performed on one human genome, and a researcher wishes to perform the same computation on another human genome, most of the work has already been done.[22] When dealing with redundant data, clustering comes to mind. While cluster-based search is well studied,[20] conventional wisdom holds that it provides a constant factor speed-up over exhaustive search.

Beyond redundancy, however, another attribute of large biological datasets stands out. Far fewer biological sequences exist than could be enumerated, but even more so, those that exist tend to be highly similar to many others. Thanks to evolution, only those genes that exhibit useful biological function survive, and most random sequences of amino acids would not be expected to form stable structures. Since two human genomes differ on average by only 0.1%, a collection of 1,000 human genomes contains less than twice the unique information of a single genome.[22] Thus, not only does biological data exhibit redundancy, it also tends not to inhabit anywhere near the entire feasible space (Figure 3). It seems that physical laws—in this case, evolution—constrain the data to a particular subspace of the Cartesian space.

One key insight related to redundancy is that such datasets exhibit low *metric entropy*.[38] That is, for a given cluster radius r_c and a database D, the number k of clusters needed to cover D is bounded by $N_{r_c}(D)$, the metric entropy, which is relatively small compared to $|D|$, the number of entries in the database (Figure 3). In contrast, if the points were uniformly distributed about the Cartesian space, $N_{r_c}(D)$ would be larger.

A second key insight is the biological datasets have low fractal dimension.[38] That is, within some range of radii r_1 and r_2 about an arbitrary point in the database D, the fractal dimension d is $d = \frac{(\log(n_2/n_1)}{(\log(r_2/r_1)}$, where n_1 and n_2 are the number of points within r_1 and r_2 respectively (Figure 3).

Cluster-based search, as exemplified by "compressive omics"—the use of compression to accelerate analysis—can perform approximate search within a radius r of a query q on a database D with fractal dimension d and metric entropy k at the scale r_c in time proportional to

$$O\left(\underbrace{k}_{\text{metric entropy}} + \overbrace{|B_D(q,r)|}^{\text{output size}} \underbrace{\left(\frac{r+2r_c}{r}\right)^d}_{\text{scaling factor}}\right),$$

where $B_D(q, r)$ refers to the set of points in D contained within a ball of radius r about a point q.

Given this formalization, the ratio $\frac{(|D|)}{k}$ provides an estimate of the speed-up factor for the coarse search component compared to a full linear search. The time complexity of the fine search is exponential in the fractal dimension d, which can be estimated globally by sampling the local fractal dimension over a dataset. The accompanying table provides the fractal dimension d sampled at typical query radii, as well as the ratio $\frac{(|D|)}{k}$, for nucleotide sequence, protein sequence, protein structure, and chemical compound databases.

Biological datasets exhibit redundancy, and are constrained to subspaces by physical laws; that is, the vast majority of enumerable sequences and structures do not exist because they are not advantageous (or at least, have not been selected for by evolution). This combination results in low fractal dimension and low metric entropy relative to the size of the dataset, which suggests that "compressive omics" will provide the ability for computation to scale sublinearly with massively growing data.

The Age of Compressive Algorithms

We are entering the age of compressive algorithms, which make use of this completely different paradigm for the structure of biological data. Seeking to take advantage of the redundancy inherent in genomic sequence data, Loh, Baym and Berger[22] introduced *compressive genomics*, an approach that relies on compressing data in such a way that the desired computation (such as BLAST search) can be performed in the compressed representation. Compressive genomics is based on the concept of *compressive acceleration*, which relies on a two-stage search, referred to as *coarse* and *fine* search. Coarse search is performed only on the coarse, or representative, subsequences that represent unique data. Any representative sequence within some threshold of the query is then expanded into all similar sequences it represents; the fine search is over this (typically small) subset of the original database. This approach provides orders-of-magnitude runtime improvements to BLAST nucleotide[22] and protein[12] search; these runtime improvements increase as databases grow.

The CORA read mapper[37] applies a mid-size *l*-mer based read-compression approach with a compressive indexing of the reference genome (referred to as a homology table). CORA, like caBLAST (compressively accelerated BLAST)[22] and caBLASTP,[12] accelerates existing tools (in this case, read mappers including BWA or Bowtie2) by allowing them to operate in a compressed space, and relies on a coarse and a fine phase. In contrast, short seed-clustering schemes, such as those used in Masai[33] and MrsFAST[3] conceptually differ from CORA in that those schemes aim to accelerate only the seed-to-reference matching step. Thus, there is a subsequent seed-extension step, which is substantially more costly and still needs to be performed for each read and mapping individually, even when seeds are clustered. Through its *l*-mer based read compression model, CORA is able to accelerate and achieve asymptotically sublinear scaling for both the seed-matching and seed-extension steps within coarse-mapping, which comprises the major bulk of the read-mapping computation. Traditionally, *k*-mers refer to short substrings of fixed length (often, but not necessarily, a power of two) used as "seeds" for longer sequence matches. CORA uses much longer *k*-mers (for example, 33–64 nucleotides long), and links each one to its neighbors within a small Hamming or Levenshtein distance. The term *l*-mer distinguishes these substrings from typically short *k*-mers.

In the area of metagenomic search, the recently released MICA[38] demonstrates the compressive-acceleration approach of caBLAST[22] and caBLASTP[12] is largely orthogonal to alphabet-reduction and indexing approaches. MICA applies the compressive-acceleration framework to the state-of-the-art DIAMOND,[7] using it for its "coarse search" phase and a user's choice of DIAMOND or BLASTX for its "fine search" phase; MICA demonstrates nearly order-of-magnitude run-time gains over the highly optimized DIAMOND, comparable to that of caBLASTP over BLASTP.

Compressive genomics[22] has been generalized and adapted to non-sequence spaces as well, and coined "compressive omics." One such example is chemogenomics. Applying a compressive acceleration approach, Ammolite[38] accelerates SMSD search by an average of 150x on the PubChem database. Another example is esFragBag,[38] which clusters proteins based on the cosine distance or Euclidean distance of their bag-of-words vectors, further accelerating FragBag's running time by an average of 10x.

The compressive omics approach can, in some cases, come at the cost of accuracy. However, these cases are well defined. Compressive omics never results in false positives (with respect to the naïve search technique being accelerated), because the fine search phase applies the same comparison to the candidates as the naïve approach. Furthermore, when the distance function used for comparisons is a metric—more specifically, when it obeys the triangle inequality—false negatives will also never occur. Yet, in practice, non-metric distance functions are used, such as E-values in BLAST or cosine

Metric-entropy ratio (ratio of clusters to entries in database) and fractal dimension at typical search radii for four datasets.

Metric-entropy ratio gives an estimate of the acceleration of coarse search with respect to naïve search, and as long as fractal dimension is low, coarse search should dominate total search time. NCBI's non-redundant 'NR' protein and 'NT' nucleotide sequence databases are from June 2015. Protein Data Bank (PDB) is from July 2015. PubChem is from October 2013.

Dataset	Metric-entropy ratio	Fractal dimension
Nucleotide sequences (NCBI NT)	7:1	1.5
Protein sequences (NCBI NR)	5:1	1.6
Protein structure (PDB)	10:1	2.5
Chemical structure (PubChem)	11:1	0.2

distance in esFragBag, and thus false negatives can occur. Fortunately, these error rates are low, and recall better than 90% has been demonstrated.[12,22,38]

Conclusion

The explosion of biological data, largely due to technological advances such as next-generation sequencing, presents us with challenges as well as opportunities. The promise of unlocking the secrets of diseases such as cancer, obesity, Alzheimer's, autism spectrum disorder, and many others, as well as better understanding the basic science of biology, relies on researchers' ability to analyze the growing flood of genomic, metagenomic, structural, and interactome data.

The approach of compressive acceleration,[22] and its demonstrated ability to scale with the metric entropy of the data,[38] while providing orthogonal benefits to many other useful indexing techniques, is an important tool for coping with the deluge of data. The extension of this compressive acceleration approach to metagenomics, NGS read mapping,[37] and chemogenomics suggests its flexibility. Likewise, compressive storage for these applications can be shown to scale with the information-theoretic entropy of the dataset.[38]

The field of computational biology must continue to innovate, but also to incorporate the best ideas from other areas of computer science. For example, the compressive acceleration approach bears similarity to a metric ball tree, first described in the database community over 20 years ago;[35] however, the latter does not allow one to analyze performance guarantees in terms of metric entropy and fractal dimension. Other ideas from image processing, computational geometry,[18] sublinear-time algorithms,[30] and other areas outside of biology are likely to bear fruit. It is also likely that algorithmic ideas developed within computational biology will become useful in other fields experiencing a data deluge, such as astronomy or social networks.[34]

Biological data science is unique for two primary reasons: biology itself—even molecular biology—predates the information age, and "nothing in biology makes sense except in light of evolution."[13] Not only have biologists developed a diverse array of experimental techniques, but the data derives from astoundingly complex processes that themselves are driven by evolution. It is through the development of algorithms that leverage the structure of biological data that we can make sense of biology in light of evolution.

Acknowledgments

This work is supported by the National Institutes of Health, under grant GM108348. Y.W.Y. is also supported by a Hertz Fellowship.

References

1. 1000 Genomes Project Consortium et al. An integrated map of genetic variation from 1,092 human genomes. *Nature 491*, 7422 (2012), 56–65.
2. Altschul, S.F., Gish, W., Miller, W., Myers, E.W. and Lipman, D.J. Basic local alignment search tool. *Journal of Molecular Biology 215*, 3 (1990), 403–410.
3. Berger, B., Peng, J. and Singh, M. Computational solutions for omics data. *Nature Reviews Genetics 14*, 5 (2013), 333–346.
4. Bonfield, J.K. and Mahoney, M.V. Compression of FASTQ and SAM format sequencing data. *PLoS ONE 8*, 3 (2013), e59190.
5. Bredel, M. and Jacoby, E. Chemogenomics: An emerging strategy for rapid target and drug discovery. *Nature Reviews Genetics 5*, 4 (2004), 262–275.
6. Bruijn, D.N. A combinatorial problem. In *Proceedings of the Koninklijke Nederlandse Akademie van Wetenschappen, Series A 49*, 7 (1946), 758.
7. Buchfink, B., Xie, C., and Huson, D.H. Fast and sensitive protein alignment using DIAMOND. *Nature Methods 12*, 1 (2015), 59–60.
8. Candes, E.J. and Tao, T. Decoding by linear programming. *IEEE Transactions on Information Theory 51*, 12 (2005), 4203–4215.
9. Cao, M., Zhang, H., Park, J., Daniels, N.M., Crovella, M.E., Cowen, L.J. and Hescott, B. Going the distance for protein function prediction: A new distance metric for protein interaction networks. *PLoS ONE 8*, 10 (2013).
10. Chindelevitch, L., Trigg, J., Regev, A. and Berger, B. An exact arithmetic toolbox for a consistent and reproducible structural analysis of metabolic network models. *Nature Communications 5*, (2014).
11. Cho, H., Berger, B., and Peng, J. Diffusion component analysis: Unraveling functional topology in biological networks. *Research in Computational Molecular Biology*. Springer, 2015, 62–64.
12. Daniels, N.M., Gallant, A., Peng, J., Cowen, L.J., Baym, M. and Berger, M. Compressive genomics for protein databases. *Bioinformatics 29* (2013), i283–i290.
13. Dobzhansky, T. Nothing in biology makes sense except in the light of evolution (1973).
14. Forsberg, K.J., Reyes, A., Wang, B., Selleck, E.M., Sommer, M.O. and Dantas, G. The shared antibiotic resistome of soil bacteria and human pathogens. *Science 337*, 6098 (2012), 1107–1111.
15. Hach, F., Hormozdiari, F., Alkan, C., Hormozdiari, F., Birol, I., Eichler, E.E. and Sahinalp, S.C. mrsFAST: A cache-oblivious algorithm for short-read mapping. *Nature Methods 7*, 8 (2010), 576–577.
16. Hach, F., Sarra, I. Hormozdiari, F., Alkan, C., Eichler, E.E. and Sahinalp, S.C. mrsFAST-Ultra: a compact, SNP-aware mapper for high-performance sequencing applications. *Nucleic Acids Research* (2014), gku370.
17. Hart, Y., Sheftel, H., Hausser, J., Szekely, P., Ben-Moshe, N.B., Korem, Y., Tendler, A., Mayo, A.E. and Alon, U. Inferring biological tasks using Pareto analysis of high-dimensional data. *Nature Methods 12*, 3 (2015), 233–235.
18. Indyk, P. and Motwani, R. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the 13th Annual ACM Symposium on Theory of Computing*. ACM, 1998, 604–613.
19. Janda, J.M. and Abbott, S.L. 16S rRNA gene sequencing for bacterial identification in the diagnostic laboratory: pluses, perils, and pitfalls. *J. Clinical Microbiology 45*, 9 (2007), 2761–2764.
20. Jardine, N. and van Rijsbergen, C.J. The use of hierarchic clustering in information retrieval. *Information Storage and Retrieval 7*, 5 (1971) 217–240.
21. Liao, C.-S., Lu, K., Baym, M., Singh, R. and Berger, B. IsoRankN: Spectral methods for global alignment of multiple protein networks. *Bioinformatics 12* (2009), i253–i258.
22. Loh, P.-R., Baym, M., and Berger, B. Compressive genomics. *Nature Biotechnology 30*, 7 (2012), 627–630.
23. MacFabe, D.F. Short-chain fatty acid fermentation products of the gut microbiome: Implications in autism spectrum disorders. *Microbial Ecology in Health and Disease 23* (2012).
24. Marco-Sola, S., Sammeth, M., Guigó, R. and Ribeca, P. The gem mapper: Fast, accurate and versatile alignment by filtration. *Nature Methods 9*, 12 (2012), 1185–1188.
25. Marx, V. Biology: The big challenges of big data. *Nature 498*, 7453 (2013), 255–260.
26. Ochoa, I., Asnani, H., Bharadia, D., Chowdhury, M., Weissman, T. and Yona, G. QualComp: A new lossy compressor for quality scores based on rate distortion theory. *BMC bioinformatics 14*, 1 (2013), 187.
27. Patro, R. and Kingsford, C. Data-dependent bucketing improves reference-free compression of sequencing reads. *Bioinformatics* (2015).
28. Prat, Y., Fromer, M., Linial, N. and Linial, M. Recovering key biological constituents through sparse representation of gene expression. *Bioinformatics 5* (2011), 655–661.
29. Rahman, S.A., Bashton, M., Holliday, G.L., Schrader, R. and Thornton, J.M. Small molecule subgraph detector (SMSD) toolkit. *J. Cheminformatics 1*, 1 (2009), 1–13.
30. Rubinfeld, R. and Shapira, A. Sublinear time algorithms. *SIAM J. Discrete Mathematics 25*, 4 (2011), 1562–1588.
31. Schatz, M.C., Langmead, B. and Salzberg, S.L. Cloud computing and the DNA data race. *Nature Biotechnology 28*, 7 (2010), 691–693.
32. Singh, R., Xu, J. and Berger, B. Global alignment of multiple protein interaction networks with application to functional orthology detection. In *Proceedings of the National Academy of Sciences 105*, 35 (2008), 12763–12768.
33. Siragusa, E., Weese, D. and Reinert, K. Fast and accurate read mapping with approximate seeds and multiple backtracking. *Nucleic Acids Research 41*, 7 (2013), e78.
34. Stephens, Z.D. et al. Big data: Astronomical or genomical? *PLoS Biol. 13*, 7 (2015), e1002195.
35. Uhlmann, J.K. Satisfying general proximity/similarity queries with metric trees. *Information Processing Letters 40*, 4 (1991), 175–179.
36. Weinstein, J.N. et al. The cancer genome atlas pan-cancer analysis project. *Nature Genetics 45*, 10 (2013), 1113–1120.
37. Yorukoglu, D., Yu, Y.W., Peng, J. and Berger, B. Compressive mapping for next-generation sequencing *Nature Biotechnology 4* (2016), 374–376.
38. Yu, Y.W., Daniels, N., Danko, D.C. and Berger, B. Entropy-scaling search of massive biological data. *Cell Systems 1*, 2 (2015), 130–140.
39. Yu, Y.W., Yorukoglu, D., Peng, J. and Berger, B. Quality score compression improves genotyping accuracy. *Nature Biotechnology 33*, 3 (2015), 240–243.
40. Zhao, Y., Tang, H. and Ye, Y. RAPSearch2: A fast and memory-efficient protein similarity search tool for next-generation sequencing data. *Bioinformatics 28*, 1 (2012), 125–126.

Bonnie Berger (bab@mit.edu) is a professor in CSAIL and the Department of Mathematics and EECS at Massachusetts Institute of Technology, Cambridge, MA.

Noah M. Daniels (ndaniels@mit.edu) is a postdoctoral associate in CSAIL and Department of Mathematics, Massachusetts Institute of Technology, Cambridge, MA.

Y. William Yu (ywy@mit.edu) is a graduate student in CSAIL and Department of Mathematics, Massachusetts Institute of Technology, Cambridge, MA.

Watch the authors discuss their work in this exclusive *Communications* video. http://cacm.acm.org/videos/computational-biology-in-the-21st-century

research highlights

DOI:10.1145/2961890

To view the accompanying paper, visit doi.acm.org/10.1145/2958738 rh

Technical Perspective
Toward Reliable Programming for Unreliable Hardware

By Todd Millstein

"IT'S NOT A bug; it's a feature!" Though this sentence is often meant as a joke, sometimes a bug really *is* a feature—when the benefits of tolerating the bug outweigh its negative impact on applications.

Designers of emerging hardware architectures are taking this point of view in order to increase energy efficiency, which is a critical concern across the computing landscape, from tiny embedded devices to enormous datacenters. Techniques such as a low-voltage mode for data-processing components and a low refresh rate for memory components can significantly decrease energy consumption. But they also increase the likelihood of *soft errors*, which are transient hardware faults that can cause an erroneous value to be computed or retrieved from memory.

Ultimately, whether these techniques should be considered bugs or features rests on the ability of software systems, and their developers, to tolerate the increase in soft errors. Fortunately, a large class of applications known as *approximate computations* is naturally error-tolerant. A book recommendation system approximates an unknown "ideal" recommendation function, for example, by clustering users with similar tastes. With enough users and data about these users, sporadic errors in the clustering computation are unlikely to cause noticeably worse recommendations. Similarly, an audio encoder can likely tolerate sporadic errors that introduce additional noise without affecting the user experience.

Even so, no application can tolerate an unbounded number of errors. At some point the book recommendations will be random and the music will be unlistenable. How can the implementers of these applications gain assurance that the quality of service will be acceptable despite the potential for soft errors?

The computing industry and research community have developed many tools and techniques for finding bugs and validating properties of programs. However, for the most part those approaches do not help to answer the question here. The issue in this setting is not whether a bug exists, but how likely the bug is to occur and how it will affect the application's behavior. Further, the bug is not in the application but rather in the underlying hardware platform. Finally, it's not even clear how to specify a desired quality-of-service level; traditional program logics based on a binary notion of truth and falsehood are not up to the task.

The following paper by Carbin et al. addresses these challenges in the context of an important subproblem. The authors introduce the notion of a *quantitative reliability specification* for a variable, which specifies a minimal acceptable probability that the variable's computed value will be correct despite the potential for soft errors. For example, a developer may desire a particular variable's value to be correct 99% of the time. The authors introduce a language for providing such specifications as well as an automated code analysis to verify them. Separately, the authors and other researchers have tackled complementary problems, such as how to bound the maximum effect that soft errors can have on a variable's value.

The power of the authors' approach comes from its generality. Despite my example here, reliability specifications are relative rather than absolute. For example, the reliability specification for a function's return value is defined in terms of the reliability probabilities of the function's arguments and so must hold for all possible values of those probabilities. Further, the approach is parameterized by a separate hardware reliability specification that provides specific probabilities of soft errors for different operations (for example, reading from memory, performing an addition). Therefore the approach is oblivious to the particular details of the hardware architecture and the causes of its soft errors.

These choices not only make the approach more general; they also enable the authors to recast the problem in a manner that is surprisingly amenable to traditional program verification techniques. Their analysis validates reliability specifications by determining the probability that each variable's computation incurs no soft errors, since that is a lower bound on the variable's probability of being reliable. By abstracting away the specific reliability probabilities of function inputs as well as of individual operations, the problem essentially becomes one of counting the number of operations that can incur soft errors and that can affect a variable's value, a task that is well suited to automated program analysis.

This work is part of an exciting stream of recent research that adapts and extends traditional program verification techniques to reason about probabilistic properties, which are abundant in modern software systems. I am hopeful this research agenda will lead to general ways of building robust systems out of potentially unreliable parts, where the notion of unreliability is broadly construed—not only soft errors, but also faulty sensor and other environmental inputs, untrusted libraries, and approximate computations themselves. The more tools we have to reason about unreliability, the more bugs we can turn into features. C

Todd Millstein is a professor of computer science at UCLA, Los Angeles, CA.

DOI:10.1145/2958738

Verifying Quantitative Reliability for Programs that Execute on Unreliable Hardware

By Michael Carbin, Sasa Misailovic, and Martin C. Rinard

Abstract

Emerging high-performance architectures are anticipated to contain unreliable components that may exhibit *soft errors*, which silently corrupt the results of computations. Full detection and masking of soft errors is challenging, expensive, and, for some applications, unnecessary. For example, approximate computing applications (such as multimedia processing, machine learning, and big data analytics) can often naturally tolerate soft errors.

We present Rely, a programming language that enables developers to reason about the quantitative reliability of an application—namely, the probability that it produces the correct result when executed on unreliable hardware. Rely allows developers to specify the reliability requirements for each value that a function produces.

We present a static quantitative reliability analysis that verifies quantitative requirements on the reliability of an application, enabling a developer to perform sound and verified reliability engineering. The analysis takes a Rely program with a reliability specification and a hardware specification that characterizes the reliability of the underlying hardware components and verifies that the program satisfies its reliability specification when executed on the underlying unreliable hardware platform. We demonstrate the application of quantitative reliability analysis on six computations implemented in Rely.

1. INTRODUCTION

Reliability is a major concern in the design of computer systems. The current goal of delivering systems with negligible error rates restricts the available design space and imposes significant engineering costs. And as other goals such as energy efficiency, circuit scaling, and new features and functionality continue to grow in importance, maintaining even current error rates will become increasingly difficult.

In response to this situation, researchers have developed numerous techniques for detecting and masking errors in both hardware[10] and software.[9,23,24] Because these techniques typically come at the price of increased execution time, increased energy consumption, or both, they can substantially hinder or even cripple overall system performance.

Many computations, however, can easily tolerate occasional errors. An *approximate computation* (including many multimedia, financial, machine learning, and big data analytics applications) can often acceptably tolerate occasional errors in its execution and/or the data that it manipulates.[7, 20, 25] A *checkable computation* can be augmented with an efficient checker that verifies either the exact correctness[4, 14] or the approximate acceptability[1] of the results that the computation produces. If the checker does detect an error, it can re-execute the computation to obtain an acceptable result.

For both approximate and checkable computations, operating without (or with at most selectively applied) mechanisms that detect and mask errors can produce (1) fast and energy efficient execution that (2) delivers acceptably accurate results often enough to satisfy the needs of their users.

1.1. Background

Approximate computations have emerged as a major component of many computing environments. Motivated in part by the observation that approximate computations can often acceptably tolerate occasional computation and/or data errors,[7, 20, 25] researchers have developed a range of new mechanisms that forgo exact correctness to optimize other objectives. Typical goals include maximizing program performance subject to an accuracy constraint and altering program execution to recover from otherwise fatal errors.[26]

Software Techniques: Most software techniques deploy *unsound transformations*—transformations that change the semantics of an original exact program. Proposed mechanisms include skipping tasks,[16, 25] loop perforation (skipping iterations of time-consuming loops),[20, 29] sampling reduction inputs,[30] multiple selectable implementations of a given component or components,[2, 3, 12, 30] dynamic knobs (configuration parameters that can be changed as the program executes)[12] and synchronization elimination (forgoing synchronization not required to produce an acceptably accurate result).[16, 18] The results show that aggressive techniques such as loop perforation can deliver up to a fourfold performance improvement with acceptable changes in the quality of the results that the application delivers.

Hardware Techniques: The computer architecture community has begun to investigate new designs that improve performance by breaking the traditional fully reliable digital abstraction that computer hardware has traditionally sought to provide. The goal is to reduce the cost of implementing a reliable abstraction on top of physical materials and manufacturing methods that are inherently unreliable. For

The original version of this paper appeared in *Proceedings of the 28th ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications* (Indianapolis, IN, Oct. 2013).

example, researchers are investigating designs that incorporate aggressive device and voltage scaling techniques to provide low-power ALUs and memories. A key aspect of these components is that they forgo traditional correctness checks and instead expose timing errors and bitflips with some non-negligible probability.[9, 11, 13, 15, 21, 22, 27]

1.2. Reasoning about approximate programs

Approximate computing violates the traditional contract that the programming system must preserve the standard semantics of the program. It therefore invalidates standard paradigms and motivates new, more general, approaches to reasoning about program behavior, correctness, and acceptability.

One key aspect of approximate applications is that they typically contain *critical* regions (which must execute without error) and *approximate* regions (which can execute acceptably even in the presence of occasional errors).[7, 25] Existing systems, tools, and type systems have focused on helping developers identify, separate, and reason about the binary distinction between critical and approximate regions.[7, 11, 15, 25, 27] However, in practice, no computation can tolerate an unbounded accumulation of errors—to execute acceptably, executions of even approximate regions must satisfy some minimal requirements.

Approximate computing therefore raises a number of fundamental new research questions. For example, what is the probability that an approximate program will produce the same result as a corresponding original exact program? How much do the results differ from those produced by the original program? And is the resulting program safe and secure?

Because traditional correctness properties do not provide an appropriate conceptual framework for addressing these kinds of questions, we instead work with *acceptability properties*—the minimal requirements that a program must satisfy for acceptable use in its designated context. We identify three kinds of acceptability properties and use the following program (which computes the minimum element `min` in an `N`-element array) to illustrate these properties:

```
int min = INT_MAX ;
for (int i = 0; i < N; ++i)
     if (a[i] < min)  min = a[i];
```

Integrity Properties: Integrity properties are properties that the computation must satisfy to produce a successful result. Examples include computation-independent properties (no out of bounds accesses, null dereferences, divide by zero errors, or other actions that would crash the computation) and computation-dependent properties (e.g., the computation must return a result within a given range). One integrity property for our example program is that accesses to the array `a` must always be within bounds.

Reliability Properties: Reliability properties characterize the probability that the produced result is correct. Reliability properties are often appropriate for approximate computations executing on unreliable hardware platforms that exhibit occasional nondeterministic errors. A potential reliability property for our example program is that `min` must be the minimum element in `a[0]–a[N–1]` with probability at least 95%.

Accuracy Properties: Accuracy properties characterize how accurate the produced result must be. For example, an accuracy property might state that the transformed program must produce a result that differs by at most a specified percentage from the result that a corresponding original program produces.[19, 30] Alternatively, a potential accuracy property for our example program might require the `min` to be within the smallest `N/2` elements `a[0]–a[N–1]`. Such an accuracy property might be satisfied by, for example, a loop perforation transformation that skips `N/2–1` of the loop iterations.

In this article we focus on reliability properties for approximate computations executing on unreliable hardware platforms. In other research, we have developed techniques for reasoning about integrity properties[5, 6] and both worst-case and probabilistic accuracy properties.[5, 19, 30] We have also extended the research presented in this article to include combinations of reliability and accuracy properties.[17]

1.3. Verifying reliability (contributions)

To meet the challenge of reasoning about reliability, we present a programming language, Rely, and an associated program analysis that computes the *quantitative reliability* of the computation—that is, the probability with which the computation produces a correct result when parts of the computation execute on unreliable hardware with soft errors (independent errors that occur nondeterministically with some probability). Specifically, given a hardware specification and a Rely program, the analysis computes, for each value that the computation produces, a conservative probability that the value is computed correctly despite the possibility of soft errors.

Rely supports and is specifically designed to enable partitioning a program into *critical* regions (which must execute without error) and *approximate* regions (which can execute acceptably even in the presence of occasional errors).[7, 25] In contrast to previous approaches, which support only a binary distinction between critical and approximate regions, quantitative reliability can provide precise static probabilistic acceptability guarantees for computations that execute on unreliable hardware platforms. This article specifically presents the following contributions:

Quantitative Reliability Specifications: We present quantitative reliability specifications, which characterize the probability that a program executed on unreliable hardware produces the correct result, as a constructive method for developing applications. Quantitative reliability specifications enable developers who build applications for unreliable hardware architectures to perform sound and verified reliability engineering.

Language: We present Rely, a language that enables developers to specify reliability requirements for programs that allocate data in unreliable memory regions and use unreliable arithmetic/logical operations.

Quantitative Reliability Analysis: We present a program analysis that verifies that the dynamic semantics of a Rely program satisfies its quantitative reliability specifications. For each function in the program, the analysis computes a symbolic reliability precondition that characterizes the set of valid specifications for the function. The analysis then

verifies that the developer-provided specifications are valid according to the reliability precondition.

Case Studies: We have used the Rely implementation to develop unreliable versions of six building block computations for media processing, machine learning, and data analytics applications. These case studies illustrate how to use quantitative reliability to develop and reason about both approximate and checkable computations in a principled way.

2. EXAMPLE

Rely is an imperative language for computations over integers, floats (not presented), and multidimensional arrays. To illustrate how a developer can use Rely, Figure 1 presents a Rely-based implementation of a pixel block search algorithm derived from that in the x264 video encoder.[a]

The function `search_ref` searches a region (`pblocks`) of a previously encoded video frame to find the block of pixels that is most similar to a given block of pixels (`cblock`) in the current frame. The motion estimation algorithm uses the results of `search_ref` to encode `cblock` as a function of the identified block.

This is an approximate computation that can trade correctness for more efficient execution by approximating the search to find a block. If `search_ref` returns a block that is not the most similar, then the encoder may require more bits to encode `cblock`, potentially decreasing the video's peak signal-to-noise ratio or increasing the video's encoded size. However, previous studies on soft error injection[9] and more aggressive transformations like loop perforation[20, 29] have demonstrated that the quality of x264's final result is only slightly affected by perturbations of this computation.

[a] x264 (http://www.videolan.org/x264.html).

Figure 1. Rely code for motion estimation computation.

```
 1  #define nblocks 20
 2  #define height  16
 3  #define width   16
 4
 5  int<0.99*R(pblocks, cblock)>
 6  search_ref (
 7    int<R(pblocks)> pblocks(3) in urel,
 8    int<R(cblock)>  cblock(2) in urel)
 9  {
10    int minssd = INT_MAX,
11        minblock = -1 in urel;
12    int ssd, t, t1, t2 in urel;
13    int i = 0, j, k;
14
15    repeat nblocks  {
16      ssd = 0;
17      j = 0;
18      repeat height {
19        k = 0;
20        repeat width {
21          t1 = pblocks[i,j,k];
22          t2 = cblock[j,k];
23          t = t1 -. t2;
24          ssd = ssd +. t *. t;
25          k = k + 1;
26        }
27        j = j + 1;
28      }
29
30      if (ssd <. minssd) {
31        minssd = ssd;
32        minblock = i;
33      }
34
35      i = i + 1;
36    }
37    return minblock;
38  }
```

2.1. Reliability specifications

The function declaration on Line 6 specifies the types and reliabilities of `search_ref`'s parameters and return value. The parameters of the function are `pblocks(3)`, a three-dimensional array of pixels, and `cblock(2)`, a two-dimensional array of pixels. In addition to the standard signature, the function declaration contains *reliability specifications* for each result that the function produces.

Rely's reliability specifications express the reliability of a function's results—when executed on an unreliable hardware platform—as a function of the reliabilities of its inputs. A reliability specification has the form $r \cdot \mathcal{R}(X)$, where r is a real number between 0 and 1 and X is a set of variables. For example, the specification for the reliability of `search_ref`'s result is `int<0.99*R(pblocks,cblock)>`. This specification states that the return value is an integer with a reliability that is at least 99% of the *joint reliability* of the parameters `pblocks` and `cblock` (denoted by `R(pblocks, cblock)`). The joint reliability of a set of parameters is the probability that they all have the correct value when passed in from the caller. This specification holds for all possible values of the joint reliability of `pblocks` and `cblock`. For instance, if the contents of the arrays `pblocks` and `cblock` are fully reliable (correct with probability one), then the return value is correct with probability 0.99.

In Rely, arrays are passed by reference and the execution of a function can, as a side effect, modify an array's contents. The reliability specification of an array therefore allows a developer to constrain the *reliability degradation* of its contents. Here `pblocks` has an output reliability specification of `R(pblocks)` (and similarly for `cblock`), meaning that all of `pblock`'s elements are at least as reliable when the function exits as they were on entry to the function.

Joint reliabilities serve as an abstraction of a function's input distribution, which enables Rely's analysis to be both modular and oblivious to the exact shape of the distributions. This is important because (1) such exact shapes can be difficult for developers to identify and specify and (2) known tractable classes of probability distributions are not closed under many operations found in standard programming languages, which can complicate attempts to develop compositional analyses that work with such exact shapes.[19, 28]

2.2. Unreliable computation

Rely targets hardware architectures that expose both reliable operations (which always execute correctly) and more

energy-efficient unreliable operations (which execute correctly with only some probability). Specifically, Rely supports reasoning about reads and writes of unreliable memory regions and unreliable arithmetic/logical operations.

Memory Region Specification: Each parameter declaration specifies the memory region in which the data of the parameter is allocated. Memory regions correspond to the physical partitioning of memory at the hardware level into regions of varying reliability. Here `pblocks` and `cblock` are allocated in an unreliable memory region named `urel`.

Lines 10–13 declare the local variables of the function. By default, variables in Rely are allocated in a fully reliable memory region. However, a developer can also optionally specify a memory region for each local variable. For example, the variables declared on Lines 10–12 reside in `urel`.

Unreliable Operations: The operations on Lines 23, 24, and 30 are unreliable arithmetic/logical operations. In Rely, every arithmetic/logical operation has an unreliable counterpart that is denoted by suffixing a period after the operation symbol. For example, "`-.`" denotes unreliable subtraction and "`<.`" denotes unreliable comparison.

Using these operations, `search_ref`'s implementation *approximately* computes the index (`minblock`) of the most similar block, that is, the block with the minimum distance from `cblock`. The `repeat` statement on line 15, iterates a constant `nblock` number of times, enumerating over all previously encoded blocks. For each encoded block, the `repeat` statements on lines 18 and 20 iterate over the `height * width` pixels of the block and compute the sum of the squared differences (`ssd`) between each pixel value and the corresponding pixel value in the current block `cblock`. Finally, the computation on lines 30 through 33 selects the block that is—approximately—the most similar to `cblock`.

2.3. Hardware semantics

Figure 2 illustrates the conceptual machine model behind Rely's reliable and unreliable operations; the model consists of a CPU and a memory.

CPU: The CPU consists of (1) a register file, (2) arithmetic logical units that perform operations on data in registers, and (3) a control unit that manages the program's execution.

The arithmetic-logical unit can execute reliably or unreliably. Figure 2 presents physically separate reliable and unreliable functional units, but this distinction can be achieved through other mechanisms, such as dual-voltage architectures.[11] Unreliable functional units may omit additional checking logic, enabling the unit to execute more efficiently but also allowing for soft errors that may occur due to, for example, power variations within the ALU's combinatorial circuits or particle strikes.

To prevent the execution from taking control flow edges that are not in the program's static control flow graph, the control unit of the CPU reliably fetches, decodes, and schedules instructions (as is supported by existing unreliable processor architectures[11, 27]). In addition, given a virtual address in the application, the control unit correctly computes a physical address and operates only on that address.

Memory: Rely supports machines with memories that consist of an arbitrary number of memory partitions (each potentially of different reliability), but for simplicity Figure 2 partitions memory into two regions: reliable and unreliable. Unreliable memories can, for example, use decreased DRAM refresh rates to reduce power consumption at the expense of increased soft error rates.[15, 27]

2.4. Hardware reliability specification

Rely's analysis works with a hardware reliability specification that specifies the reliability of arithmetic/logical and memory operations. Figure 3 presents a hardware reliability specification that is inspired by results from the existing computer architecture literature.[10, 15] Each entry specifies the reliability—the probability of a correct execution—of arithmetic operations (e.g., `+.`) and memory read/write operations.

For ALU operations, the presented reliability specification uses the reliability of an unreliable multiplication operation from Ref.[10], Figure 9. For memory operations, the specification uses the probability of a bit flip in a memory cell from Ref.[15], Figure 4 with extrapolation to the probability of a bit flip within a 32-bit word. Note that a memory region specification includes two reliabilities: the reliability of a read (`rd`) and the reliability of a write (`wr`).

Figure 2. Machine model. Orange boxes represent unreliable components.

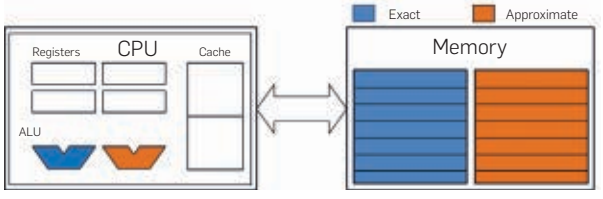
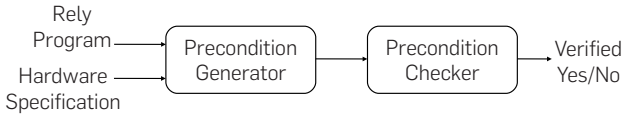


Figure 3. Hardware reliability specification.

```
reliability spec {
  operator (+.) = 1 - 10^-7;
  operator (-.) = 1 - 10^-7;
  operator (*.) = 1 - 10^-7;
  operator (<.) = 1 - 10^-7;
  memory rel {rd = 1, wr = 1};
  memory urel {rd = 1 - 10^-7, wr = 1};
}
```

Figure 4. Rely analysis overview.



2.5. Reliability analysis

Given a Rely program, Rely's reliability analysis verifies that the each function in the program satisfies its reliability specification when executed on unreliable hardware. Figure 4 presents an overview of Rely's analysis. The analysis consists of two components: the *precondition generator* and the *precondition checker*.

Precondition Generator: Given a Rely program and a hardware reliability specification, the precondition generator generates a symbolic *reliability precondition* for each function. A reliability precondition is a set of constraints that is sufficient to ensure that a function satisfies its reliability specification when executed on the underlying unreliable hardware platform. The reliability precondition is a conjunction of predicates of the form $A_{out} \leq r \cdot \mathcal{R}(X)$, where A_{out} is a placeholder for a developer-provided reliability specification for an output named *out*, r is a real number between 0 and 1, and the term $\mathcal{R}(X)$ is the joint reliability of a set of parameters X.

Conceptually, each predicate specifies that the reliability given in the specification (given by A_{out}) should be less than or equal to the reliability of a path that the program may take to compute the result (given by $r \cdot \mathcal{R}(X)$). The analysis computes the reliability of a path from the probability that all operations along the path execute reliably.

The specification is valid if the probabilities for all paths to computing a result exceed that of the result's specification. To avoid the inherent intractability of considering all possible paths, Rely uses a simplification procedure to reduce the precondition to one that characterizes the least reliable path(s) through the function.

Precondition Checker: Rely verifies that the function's specifications are consistent with its reliability precondition. Because reliability specifications are also of the form $r \cdot \mathcal{R}(X)$, the final precondition is a conjunction of predicates of the form $r_1 \cdot \mathcal{R}(X_1) \leq r_2 \cdot \mathcal{R}(X_2)$, where $r_1 \cdot \mathcal{R}(X_1)$ is a reliability specification and $r_2 \cdot \mathcal{R}(X_2)$ is a path reliability. If these predicates are valid, then the reliability of each computed output is greater than that given by its specification.

The validity problem for these predicates has a sound mapping to the conjunction of two simple constraint validity problems: inequalities between real numbers ($r_1 \leq r_2$) and set inclusion constraints over finite sets ($X_2 \subseteq X_1$). Checking the validity of a reliability precondition is therefore decidable and efficiently checkable.

Design: As a key design point, the analysis generates preconditions according to a conservative approximation of the semantics of the function. Specifically, it characterizes the reliability of a function's result according to the probability that the function computes that result fully reliably.

To illustrate the intuition behind this design point, consider the evaluation of an integer expression e. The reliability of e is the probability that it evaluates to the same value n in an unreliable evaluation as in the fully reliable evaluation. There are two ways that an unreliable evaluation can return n: (1) the unreliable evaluation of e encounters no faults and (2) the unreliable evaluation possibly encounters faults, but still returns n by chance.

Rely's analysis conservatively approximates the reliability of a computation by only considering the first scenario. This design point simplifies the reasoning to the task of computing the probability that a result is reliably computed as opposed to reasoning about a computation's input distribution and the probabilities of all executions that produce the correct result. As a consequence, the analysis requires as input only a hardware reliability specification that gives the probability with which each arithmetic/logical operation and memory operation executes correctly. The analysis is therefore oblivious to a computation's input distribution and does not require a full model of how soft errors affect its result.

Precondition generator. For each function, Rely's analysis generates a reliability precondition that conservatively bounds the set of valid specifications for the function. The analysis produces this precondition by starting at the end of the function from a postcondition that must be true when the function returns and then working backward to produce a precondition such that if the precondition holds before execution of the function, then the postcondition holds at the end of the function.

Postcondition: The postcondition for a function is the constraint that the reliability of each array argument exceeds that given in its specification. For `search_ref`, the postcondition Q_0 is

$$Q_0 = A_{\texttt{pblocks}} \leq \mathcal{R}(\texttt{pblocks}) \wedge A_{\texttt{cblock}} \leq \mathcal{R}(\texttt{cblock}),$$

which specifies that the reliability of the arrays `pblocks` and `cblock`—$\mathcal{R}$(`pblocks`) and $\mathcal{R}$(`cblock`)—should be at least that specified by the developer—$A_{\texttt{pblocks}}$ and $A_{\texttt{cblock}}$.

Precondition Generation: The analysis of the body of the `search_ref` function starts at the `return` statement. Given the postcondition Q_0, the analysis creates a new precondition Q_1 by conjoining to Q_0 a predicate that states that the reliability of the return value ($r_0 \cdot \mathcal{R}$(`minblock`)) is at least that of its specification (A_{ret}):

$$Q_1 = Q_0 \wedge A_{ret} \leq r_0 \cdot \mathcal{R}(\texttt{minblock}).$$

The reliability of the return value comes from the design principle for reliability approximation. Specifically, this reliability is the probability of correctly reading `minblock` from unreliable memory—which is $r_0 = 1 - 10^{-7}$ according to the hardware reliability specification—multiplied by $\mathcal{R}$(`minblock`), the probability that the preceding computation correctly computed and stored `minblock`.

Loops: The statement that precedes the `return` statement is the `repeat` statement on Line 15. A key difficulty with reasoning about the reliability of variables modified within a loop is that if a variable is updated unreliably and has a loop-carried dependence then its reliability monotonically decreases as a function of the number of loop iterations. Because the reliability of such variables can, in principle, decrease arbitrarily in an unbounded loop, Rely provides both an unbounded loop statement (with an associated analysis) and an alternative *bounded loop* statement that lets a developer specify a compile-time bound

on the maximum number of its iterations that therefore bounds the reliability degradation of modified variables. The loop on Line 15 iterates `nblocks` times and therefore decreases the reliability of any modified variables `nblocks` times. Because the reliability degradation is bounded, Rely's analysis uses unrolling to reason about the effects of a bounded loop.

Conditionals: The analysis of the body of the loop on Line 15 encounters the `if` statement on Line 30.[b] This `if` statement uses an unreliable comparison operation on `ssd` and `minssd`, both of which reside in unreliable memory. The reliability of `minblock` when modified on Line 32 therefore also depends on the reliability of this expression because faults may force the execution down a different path.

Figure 5 presents a Hoare logic style presentation of the analysis of the conditional statement. The analysis works in three steps; the preconditions generated by each step are numbered with the corresponding step.

Step 1: To capture the implicit dependence of a variable on an unreliable condition, Rely's analysis first uses latent *control flow variables* to make these dependencies explicit. A control flow variable is a unique program variable (one for each statement) that records whether the conditional evaluated to *true* or *false*. We denote the control flow variable for the `if` statement on Line 30 by ℓ_{30}.

To make the control flow dependence explicit, the analysis adds the control flow variable to all joint reliability terms in Q_1 that contain variables modified within the body of the `if` conditional (`minssd` and `minblock`).

Step 2: The analysis next recursively analyzes both the "then" and "else" branches of the conditional, producing one precondition for each branch. As in a standard precondition generator (e.g., weakest-preconditions) the assignment of `i` to `minblock` in the "then" branch replaces `minblock` with `i` in the precondition. Because reads from `i` and writes to `minblock` are reliable (according to the specification) the analysis does not introduce any new r_0 factors.

[b] This happens after encountering the increment of `i` on Line 35, which does not modify the current precondition because it does not reference `i`.

Figure 5. if statement analysis in the last loop iteration.

```
(3)  {Q0 ∧ Aret ≤ r0^4 · R(i, ssd, minssd)
        ∧ Aret ≤ r0^4 · R(minblock, ssd, minssd)}
     if (ssd <. minssd) {
(2)     {Q0 ∧ Aret ≤ r0 · R(i, ℓ30)}
        minssd = ssd;
        {Q0 ∧ Aret ≤ r0 · R(i, ℓ30)}
        minblock = i;
        {Q0 ∧ Aret ≤ r0 · R(minblock, ℓ30)}
     } else {
(2)     {Q0 ∧ Aret ≤ r0 · R(minblock, ℓ30)}
        skip;
        {Q0 ∧ Aret ≤ r0 · R(minblock, ℓ30)}
     }
(1)  {Q0 ∧ Aret ≤ r0 · R(minblock, ℓ30)}
```

Step 3: In the final step, the analysis leaves the scope of the conditional and conjoins the two preconditions for its branches after transforming them to include the direct dependence of the control flow variable on the reliability of the `if` statement's condition expression.

The reliability of the `if` statement's expression is greater than or equal to the product of (1) the reliability of the `<.` operator (r_0), (2) the reliability of reading both `ssd` and `minssd` from unreliable memory (r_0^2), and (3) the reliability of the computation that produced `ssd` and `minssd` ($\mathcal{R}$(`ssd`, `minssd`)). The analysis therefore transforms each predicate that contains the variable ℓ_{30}, by multiplying the right-hand side of the inequality with r_0^3 and replacing the variable ℓ_{30} with `ssd` and `minssd`.

This produces the precondition Q_2:

$$Q_2 = Q_0 \wedge A_{ret} \le r_0^4 \cdot \mathcal{R}(\texttt{i}, \texttt{ssd}, \texttt{minssd}) \wedge A_{ret} \le r_0^4 \cdot \mathcal{R}(\texttt{minblock}, \texttt{ssd}, \texttt{minssd}).$$

Simplification: After unrolling a single iteration of the loop that begins at Line 15, the analysis produces $Q_0 \wedge A_{ret} \le r_0^{2564} \cdot \mathcal{R}$(`pblocks`, `cblock`, `i`, `ssd`, `minssd`) as the precondition for a single iteration of the loop's body. The constant 2564 represents the number of unreliable operations within a single loop iteration.

Note that there is one less predicate in this precondition than in Q_2. As the analysis works backwards through the program, it uses a simplification technique that identifies that a predicate $A_{ret} \le r_1 \cdot \mathcal{R}(X_1)$ *subsumes* another predicate $A_{ret} \le r_2 \cdot \mathcal{R}(X_2)$. Specifically, the analysis identifies that $r_1 \le r_2$ and $X_2 \subseteq X_1$, which together mean that the second predicate is a weaker constraint on A_{ret} than the first and can therefore be removed. This follows from the fact that the joint reliability of a set of variables is less than or equal to the joint reliability of any subset of the variables—regardless of the distribution of their values.

This simplification is how Rely's analysis achieves scalability when there are multiple paths in the program; specifically a simplified precondition characterizes the least reliable path(s) through the program.

Final Precondition: When the analysis reaches the beginning of the function after fully unrolling the loop on Line 15, it has a precondition that bounds the set of valid specifications as a function of the reliability of the parameters of the function. For `search_ref`, the analysis generates the precondition

$$A_{ret} \le 0.994885 \cdot \mathcal{R}(\texttt{pblocks}, \texttt{cblock}) \wedge A_{\texttt{pblocks}} \le \mathcal{R}(\texttt{pblocks}) \wedge A_{\texttt{cblock}} \le \mathcal{R}(\texttt{cblock}).$$

Precondition checker. The final precondition is a conjunction of predicates of the form $A_{out} \le r \cdot \mathcal{R}(X)$, where A_{out} is a placeholder for the reliability specification of an output. Because reliability specifications are all of the form $r \cdot \mathcal{R}(X)$, each predicate in the final precondition (where each A_{out} is replaced with its specification) is of the form form $r_1 \cdot \mathcal{R}(X_1) \le r_2 \cdot \mathcal{R}(X_2)$, where $r_1 \cdot \mathcal{R}(X_1)$ is a reliability specification and $r_2 \cdot \mathcal{R}(X_2)$ is computed by the analysis. Similar to the analysis's simplifier (see Precon-

dition checker section), the precondition checker verifies the validity of each predicate by checking that (1) r_1 is less than r_2 and (2) $X_2 \subseteq X_1$.

For `search_ref`, the analysis computes the following predicates:

$$0.99 \cdot \mathcal{R}(\texttt{pblocks}, \texttt{cblock}) \leq 0.994885 \cdot \mathcal{R}(\texttt{pblocks}, \texttt{cblock})$$
$$\mathcal{R}(\texttt{pblocks}) \leq \mathcal{R}(\texttt{pblocks})$$
$$\mathcal{R}(\texttt{cblock}) \leq \mathcal{R}(\texttt{cblock}).$$

Because these predicates are valid according to the checking procedure, `search_ref` satisfies its reliability specification when executed on the specified unreliable hardware.

3. CASE STUDIES

We have used Rely to build unreliable versions of six building block computations for media processing, machine learning, and data analytics applications. These case studies illustrate how quantitative reliability enables a developer to use principled reasoning to relax the semantics of both *approximate computations* and *checkable computations*.

Benchmarks: We analyze the following six computations:

- **newton:** This computation searches for a root of a univariate function using Newton's Method.
- **bisect:** This computation searches for a root of a univariate function using the Bisection Method.
- **coord:** This computation calculates the Cartesian coordinates from the polar coordinates passed as the input.
- **search_ref:** This computation performs a simple motion estimation. We presented this computation in Section 2.
- **mat_vec:** This computation multiplies a matrix and a vector and stores the result in another vector.
- **hadamard:** This computation takes as input two blocks of 4×4 pixels and computes the sum of differences between the pixels in the frequency domain.

3.1. Deriving reliability specification

A developer's choice of reliability specifications is typically influenced by the perceived effect that the unreliable execution of the computation may have on the accuracy of the full program's result and its execution time and energy consumption. We present two strategies for how developers can use Rely to reason about the trade-offs between accuracy and performance that are available for checkable computations and approximate computations.

Checkable Computations: Checkable computations can be augmented with an efficient checker that dynamically verifies the correctness of the computation's result. If the checker detects an error, then it re-executes the computation or executes an alternative reliable implementation. For instance, a Newton's method computation searches for value of input x for which a function $f(x)$ is 0. Once this computation finds a zero of the function, x_0, it is typically much less expensive to compute $f(x_0)$ and check if it equals 0.

Quantitative reliability enables a developer to model the performance of this checked implementation of the computation. We will use T_{pass} to denote the expected time required to compute the correct value of the computation and perform the check and T_{fail} to denote the expected time required to compute an incorrect result, perform the check, and then rerun the reliable version of the computation (that produces the correct result).

If r denotes the reliability of the computation, then the expected execution time of the checked computation as a whole is $T' = r \cdot T_{pass} + (1 - r) \cdot T_{fail}$. This time can be compared with the time to always perform a reliable version of the computation. Therefore, this reasoning allows a developer to find the reliability r that meets the developer's performance improvement goal and can be analogously applied for alternative resource usage measures, such as energy consumption and throughput.

Approximate Computations: For computations that are inherently approximate, we can perform reliability profiling to relate the errors in the approximate computational kernels to the full application's errors.

To estimate the error of the computation, a developer can provide a *sensitivity testing procedure*, that specifies how the noise can be injected in the application. For instance, to estimate the error of the function `search_ref` from Figure 1, a profiler can modify the program to produce the correct minimum distance block with probability r and produce the maximum distance block with probability $1 - r$. This modification provides a conservative estimate of the bound on `search_ref`'s accuracy loss given the reliability r (when the computation's inputs are reliable) and the assumption that a fault causes `search_ref` to return the worst-case result.

The profiler can then run the application on representative inputs, for different values of r. The profiler then compares the outputs of the original and modified program by computing a developer-provided application level quality-loss-metric. The profiler estimates the relationship between computation-level error, controlled by r, and application-level quality loss. Based on this estimate, the developer can select an appropriate value of r. In our example, if the developer is willing to accept 1% loss in the video's peak-signal-to-noise ratio (the quality-loss metric for the video encoder), then this procedure can help the developer select r to be 0.98.

Benchmark analysis summary.

				Predicates	
Benchmark	**Type**	**LOC**	**Time (ms)**	**N**	**S**
newton	Checkable	21	8	82	1
bisect	Checkable	30	7	16,356	2
coord	Checkable	36	19	20	1
search_ref	Approximate	37	348	36,205	3
matvec	Approximate	32	110	1061	4
hadamard	Approximate	87	18	3	3

3.2. Analysis summary

The table here presents Rely's analysis results on the benchmark computations. For each benchmark, the table presents the type of the computation (checkable or approximate), its length in lines of code (LOC), the execution time of the analysis, and the number of inequality predicates in the final precondition produced by the precondition generator both without and with our simplification strategy.

Analysis Time: The analysis times for all benchmarks are under one second when executed on an Intel Xeon E5520 machine with 16 GB of main memory.

Number of Predicates: We used Rely with the hardware reliability specification from Figure 3 to generate a reliability precondition for each benchmark. The second to last column (labeled N) presents the number of predicates in the precondition when using a naïve strategy that does not include our simplification procedure. The rightmost column (labeled S) presents the number of predicates in each precondition when Rely employs our simplification procedure.

When Rely uses simplification, the size of each precondition is small (all consisting of less than five predicates). The difference in size between the naïvely generated preconditions and those generated via simplification demonstrates that simplification reduces the size of preconditions by multiple orders of magnitude. Simplification achieves these results by identifying that many of the additional predicates introduced by the reasoning required for conditionals can be removed. These additional predicates are often subsumed by another predicate.

4. RELATED WORK

In this section, we present an overview of the other work that intersects with Rely and its contributions to modeling and analysis of approximate computations, and computation fault tolerance.

Integrity: Almost all approximate computations have critical regions that must execute without error for the computation as a whole to execute acceptably. *Dynamic criticality analyses* automatically change different regions of the computation or internal data structures, and observe how the change affects the program's output, for example, Refs.[7, 20, 25] In addition, specification-based *static criticality analyses* let the developer identify and separate critical and approximate program regions, for example, Refs.[15, 27] Carbin et al.[5] present a verification system for relaxed approximate programs based on a relational Hoare logic. The system enables rigorous reasoning about the integrity and worst-case accuracy properties of a program's approximate regions.

In contrast to the prior static analyses that focus on the binary distinction between reliable and approximate computations, Rely allows a developer to specify and verify that even approximate computations produce the correct result *most of the time*. Overall, this additional information can help developers better understand the effects of deploying their computations on unreliable hardware and exploit the benefits that unreliable hardware offers.

Accuracy: In addition to reasoning about how often a computation may produce a correct result, it may also be desirable to reason about the accuracy of the result that the computation produces. Dynamic techniques observe the accuracy impact of program transformations, for example, Refs.,[2, 3, 16, 20, 25, 29] or injected soft errors, for example, Refs.[9, 15, 27] Researchers have developed static techniques that use probabilistic reasoning to characterize the accuracy impact of various sources of uncertainty.[8, 19, 30] And of course, the accuracy impact of the floating point approximation to real arithmetic has been extensively studied in numerical analysis.

More recently, we developed the Chisel optimization system to automate the placement of approximate operations and data.[17] Chisel extends the Rely reliability specifications (that capture acceptable frequency of errors) with absolute error specifications (that also capture acceptable magnitude of errors). Chisel formulates an integer optimization problem to automatically navigate the tradeoff space and generate an approximate computation that provides maximum energy savings (for the given model of approximate hardware) while satisfying the developer's reliability and absolute error specifications.

Fault Tolerance and Resilience: Researchers have developed various software, hardware, or mixed approaches for detection and recovery from specific types of soft errors that guarantee a reliable program execution, for example, Refs.[9, 23, 24] For example, Reis et al.[24] present a compiler that replicates a computation to detect and recover from single event upsets. These techniques are complementary to Rely in that each can provide implementations of operations that need to be reliable, as either specified by the developer or as required by Rely, to preserve memory safety and control flow integrity.

5. CONCLUSION

Driven by hardware technology trends, future computational platforms are projected to contain unreliable hardware components. To safely exploit the benefits (such as reduced energy consumption) that such unreliable components may provide, developers need to understand the effect that these components may have on the overall reliability of the approximate computations that execute on them.

We present a language, Rely, for exploiting unreliable hardware and an associated analysis that provides probabilistic reliability guarantees for Rely computations executing on unreliable hardware. By enabling developers to better understand the probabilities with which this hardware enables approximate computations to produce correct results, these guarantees can help developers safely exploit the benefits that unreliable hardware platforms offer.

Acknowledgments

We thank Deokhwan Kim, Hank Hoffmann, Vladimir Kiriansky, Stelios Sidiroglou, and Rishabh Singh for their insightful comments.

This research was supported in part by the National Science Foundation (Grants CCF-0905244, CCF-1036241, CCF-1138967, CCF-1138967, and IIS-0835652), the United States Department of Energy (Grant DE-SC0008923), and DARPA (Grants FA8650-11-C-7192, FA8750-12-2-0110). C

References

1. Achour, S., Rinard, M. Approximate checkers for approximate computations in topaz. In *OOPSLA* (2015).
2. Ansel, J., Wong, Y., Chan, C., Olszewski, M., Edelman, A., Amarasinghe, S. Language and compiler support for auto-tuning variable-accuracy algorithms. In *CGO* (2011).
3. Baek, W., Chilimbi, T.M. Green: A framework for supporting energy-conscious programming using controlled approximation. In *PLDI* (2010).
4. Blum, M., Kanna, S. Designing programs that check their work. In *STOC* (1989).
5. Carbin, M., Kim, D., Misailovic, S., Rinard, M. Proving acceptability properties of relaxed nondeterministic approximate programs. In *PLDI* (2012).
6. Carbin, M., Kim, D., Misailovic, S., Rinard, M. Verified integrity properties for safe approximate program transformations. In *PEPM* (2013).
7. Carbin, M., Rinard, M. Automatically identifying critical input regions and code in applications. In *ISSTA* (2010).
8. Chaudhuri, S., Gulwani, S., Lublinerman, R., Navidpour, S. Proving programs robust. In *FSE* (2011).
9. de Kruijf, M., Nomura, S., Sankaralingam, K. Relax: An architectural framework for software recovery of hardware faults. In *ISCA* (2010).
10. Ernst, D., Kim, N.S., Das, S., Pant, S., Rao, R., Pham, T., Ziesler, C., Blaauw, D., Austin, T., Flautner, K., Mudge, T. Razor: A low-power pipeline based on circuit-level timing speculation. In *MICRO* (2003).
11. Esmaeilzadeh, H., Sampson, A., Ceze, L., Burger, D. Architecture support for disciplined approximate programming. In *ASPLOS* (2012).
12. Hoffman, H., Sidiroglou, S., Carbin, M., Misailovic, S., Agarwal, A., Rinard, M. Dynamic knobs for responsive power-aware computing. In *ASPLOS* (2011).
13. Leem, L., Cho, H., Bau, J., Jacobson, Q., Mitra, S. Ersa: Error resilient system architecture for probabilistic applications. In *DATE* (2010).
14. Leveson, N., Cha, S., Knight, J.C., Shimeall, T. The use of self checks and voting in software error detection: An empirical study. In *IEEE TSE* (1990).
15. Liu, S., Pattabiraman, K., Moscibroda, T., Zorn, B. Flikker: Saving dram refresh-power through critical data partitioning. In *ASPLOS* (2011).
16. Meng, J., Chakradhar, S., Raghunathan, A. Best-effort parallel execution framework for recognition and mining applications. In *IPDPS* (2009).
17. Misailovic, S., Carbin, M., Achour, S., Qi, Z., Rinard, M. Chisel: Reliability- and accuracy-aware optimization of approximate computational kernels. In *OOPSLA* (2014).
18. Misailovic, S., Kim, D., Rinard, M. Parallelizing sequential programs with statistical accuracy tests. *ACM TECS Special Iss. Prob. Embedded Comput.* (2013).
19. Misailovic, S., Roy, D., Rinard, M. Probabilistically accurate program transformations. In *SAS* (2011).
20. Misailovic, S., Sidiroglou, S., Hoffmann, H., Rinard, M. Quality of service profiling. In *ICSE* (2010).
21. Narayanan, S., Sartori, J., Kumar, R., Jones, D. Scalable stochastic processors. In *DATE* (2010).
22. Palem, K. Energy aware computing through probabilistic switching: A study of limits. *IEEE Trans. Comput.* (2005).
23. Perry, F., Mackey, L., Reis, G., Ligatti, J., August, D., Walker, D. Fault-tolerant typed assembly language. In *PLDI* (2007).
24. Reis, G., Chang, J., Vachharajani, N., Rangan, R., August, D. Swift: Software implemented fault tolerance. In *CGO* (2005).
25. Rinard, M. Probabilistic accuracy bounds for fault-tolerant computations that discard tasks. In *ICS* (2006).
26. Rinard, M., Cadar, C., Dumitran, D., Roy, D., Leu, T., Beebee, W. Jr. Enhancing server availability and security through failure-oblivious computing. In *OSDI* (2004).
27. Sampson, A., Dietl, W., Fortuna, E., Gnanapragasam, D., Ceze, L., Grossman, D. EnerJ: Approximate data types for safe and general low-power computation. In *PLDI* (2011).
28. Sankaranarayanan, S., Chakarov, A., Gulwani, S. Static analysis for probabilistic programs: Inferring whole program properties from finitely many paths. In *PLDI* (2013).
29. Sidiroglou, S., Misailovic, S., Hoffmann, H., Rinard, M. Managing performance vs. accuracy trade-offs with loop perforation. In *FSE* (2011).
30. Zhu, Z., Misailovic, S., Kelner, J., Rinard, M. Randomized accuracy-aware program transformations for efficient approximate computations. In *POPL* (2012).

Michael Carbin, Sasa Misailovic, and Martin C. Rinard, Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA.

DOI:10.1145/2961892

To view the accompanying paper, visit doi.acm.org/10.1145/2958736 rh

Technical Perspective
Why Didn't I Think of That?

By Philip Wadler

ONCE IN A while, an idea will strike you with great force, and you say "Why didn't I think of that?" The history of programming the Web is a sequence of innovations in abstraction, each of which might make you utter the aforementioned phrase.

The Web is supported by an array of interlocked standards, including HTTP, HTML, CSS, CGI, and JavaScript. On top of these have been built a series of abstractions, each increasing the ease with which a Web application can be designed and implemented.

One of the earliest of these abstractions appeared at the end of the last millennium, when Atkins, Ball, Bruns, and Cox[1] devised MAWL, the Mother of All Web Languages. MAWL introduced to the Web world the now-common idea of inversion of control, where a sequence of requests from users invoking programs that generate Web pages is instead viewed as a single program generating a sequence of pages to which users respond. My first attempts to come to grips with inversion of control hurt my brain, but within a few years it acquired firm foundations in theory, relating it to well-understood notions of continuations and continuation-passing style, thanks to the efforts of Queinnec[7] and the PLT Scheme (now Racket) team of Graunke, Findler, Krishnamurthi, Van Der Hoeven, and Felleisen.[4]

Web programming was complex because it involved a plethora of programs written in different languages running on different platforms. Typically, a three-tier system consisted of JavaScript on the client, Java (or some other language) on the server, and SQL on the database. The idea of generating all three tiers from a single source was christened "programming without tiers" by Cooper, Lindley, Wadler, and Yallop.[2] Many systems independently appeared generating two or three tiers from a single source, including Google's AWT and Microsoft's LINQ, and research-oriented systems including Ocsygen, Opa, and Hop.

Advanced as these systems were, even as simple a matter as entering a date on a form could be complex. It might be input as a single string from a form that needed to be parsed, or three drop-down menus for day, month, and year that need to be assembled, or through a calendar widget in JavaScript. It was not until 2006 that I saw the iData system of Plasmeijer and Achten,[6] which suggested Web systems should abstract away from such detail, introducing model-view abstraction to Web forms, encapsulating how data was input separately from how it was to be processed. An obvious idea—but only in retrospect. Cooper, Lindley, Wadler, and Yallop[3] reworked this aspect of iData into formlets. Just as inversion of control was easier to absorb once it was related to the known notion of continuations, so formlets benefited from fitting into the known notion of applicatives, as introduced by McBride and Patterson.[5] The theory aided practice: developers wrote formlet libraries for F#, Haskell, JavaScript, and Racket, and incorporated formlet support into frameworks including Happstack, Tupil, WebSharper, and Yesod.

The following paper presents the next step. Until now, the database in a Web application has been treated as a global variable, accessible to all. Chlipala suggests a better approach: allow each module to declare locally a portion of the database relevant to its needs, and hide that portion from the other modules. He also introduces primitives to support concurrency and transaction, with a more elegant design than found in most other Web languages. Finally, he suggests a novel form of functional reactive programming that incorporates imperative actions. How the latter compares with the more declarative form of functional reactive programming found in languages such as Elm remains to be seen. Chlipala has tried these techniques in practice, and an intriguing list of his early customers may be found in the research version of this paper, which appeared in POPL 2015.

Until now, the database in a Web application has been treated as a global variable, accessible to all. Chlipala suggests a better approach.

Modularizing database access is a simple idea of enormous power, and I expect it will be coming to a Web programming language near you soon. Why didn't I think of that? C

References

1. Atkins, D.L., Ball, T., Bruns, G. and Cox, K. Mawl: A domain-specific language for form-based services. *IEEE Trans. Software Engineering 25*, 3 (1999), 334–346.
2. Cooper, E., Lindley, S., Wadler, P. and Yallop, J. Links: Web programming without tiers. *Formal Methods for Components and Objects*. Springer, 2007, 266–296.
3. Cooper, E., Lindley, S., Wadler, P. and Yallop, J. The essence of form abstraction. In *Proceedings of the Asian Symposium on Programming Languages and Systems*. Springer, 2008, 205–220.
4. Graunke, P., Krishnamurthi, S., Van Der Hoeven, S. and Felleisen, M. Programming the Web with high-level programming languages. In *Proceedings of the European Symposium on Programming*. Springer, 2001, 122–136.
5. McBride, C. and Paterson, R. Applicative programming with effects. *J. Functional Programming 18*, 1 (2008), 1–13.
6. Plasmeijer, R. and Achten, P. idata for the World Wide Web—Programming interconnected Web forms. In *Proceedings of the International Symposium on Functional and Logic Programming*. Springer, 2006, 242–258.
7. Queinnec, C. The influence of browsers on evaluators or, continuations to program Web servers. *ACM SIGPLAN Notices 35* (2000), 23–33.

Phil Wadler (wadler@inf.ed.ac.uk) is Professor of Theoretical Computer Science in the Laboratory for Foundations of Computer Science in the School of Informatics at the University of Edinburgh, Scotland.

DOI:10.1145/2958736

Ur/Web: A Simple Model for Programming the Web

By Adam Chlipala

Abstract

The World Wide Web has evolved gradually from a document delivery platform to an architecture for distributed programming. This largely unplanned evolution is apparent in the set of interconnected languages and protocols that any Web application must manage. This paper presents Ur/Web, a domain-specific, statically typed functional programming language with a much simpler model for programming modern Web applications. Ur/Web's model is *unified*, where programs in a single programming language are compiled to other "Web standards" languages as needed; supports novel kinds of *encapsulation* of Web-specific state; and exposes *simple concurrency*, where programmers can reason about distributed, multithreaded applications via a mix of transactions and cooperative preemption. We give a tutorial introduction to the main features of Ur/Web.

1. INTRODUCTION

The World Wide Web is a very popular platform today for programming certain kinds of distributed applications with graphical user interfaces (GUIs). Today's complex ecosystem of "Web standards" was not planned monolithically. Rather, it evolved gradually, from the starting point of the Web as a delivery system for static documents. The result is not surprising: there are many pain points in implementing rich functionality on top of the particular languages that browsers and servers speak. At a minimum, today's rich applications must generate HTML, for document structure; CSS, for document formatting; JavaScript, a scripting language for client-side interactivity; and messages of HTTP, a protocol for sending all of the above and more, to and from browsers. Most recent, popular applications also rely on languages like JSON for serializing complex datatypes for network communication, and on languages or APIs like SQL for storing persistent, structured data on servers. Code fragments in these different languages are often embedded within each other in complex ways, and the popular Web development tools provide little help in catching inconsistencies.

These complaints are not new, nor are language-based solutions. The Links project[4,8] pioneered the "tierless programming" approach, combining all the pieces of dynamic Web applications within one statically typed functional programming language. Similar benefits are attained in more recent mainstream designs, such as Google's Web Toolkit[a] and Closure[b] systems, for adding compilation on top of Web-standard languages; and Microsoft's LINQ,[12] for type-safe querying (to SQL databases and more) within general-purpose languages.

Such established systems provide substantial benefits to Web programmers, but there is more we could ask for. This article focuses on a language design that advances the state of the art by addressing two key desiderata. First, we bring *encapsulation* to rich Web applications, supporting program modules that treat key pieces of Web applications as private state. Second, we expose a *simple concurrency* model to programmers, while supporting the kinds of nontrivial communication between clients and servers that today's applications take advantage of. Most Web programmers seem unaware of either property as something that might be worth asking for, so part of our mission here is to evangelize for them.

We present the *Ur/Web* programming language, an extension of the Ur language,[5] a statically typed functional language inspired by dependent type theory. Open-source implementations of Ur/Web have been available since 2006, and several production Web applications use the language, including at least one profitable commercial site.

Ur/Web reduces the nest of Web standards to a simple programming model, coming close to retaining just the essence of the Web as an application platform, from the standpoints of security and performance. We have a single distributed system, with one server under the programmer's control and many clients that are not. The server and clients communicate only through various strongly typed communication channels, and every such interaction occurs as part of a transaction that appears to execute atomically, with no interference from other actions taking place at the same time. The server has access to persistent state in an SQL database, which is also accessed only through channels with strong types specific to the data schema. Clients maintain their GUIs though a novel variant of functional-reactive programming.

The next section expands on these points with a tutorial introduction to Ur/Web. We highlight the impact on the language design of our goals to support *encapsulation* and *simple concurrency*. Afterward, we compare with other research projects and widely used frameworks.

The open-source implementation of Ur/Web is available at:

http://www.impredicative.com/ur/.

2. A TUTORIAL INTRODUCTION TO UR/WEB

We will introduce the key features of Ur/Web through a series of refinements of one example, a multiuser chat application. Visitors to the site choose from a selection of chat

The original version of this paper was published in the *Proceedings of the 42nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming*. ACM, New York, NY, 2015, 153–165.

[a] http://www.gwtproject.org/.

[b] https://developers.google.com/closure/.

rooms, each of which maintains a log of messages. Any visitor to a chat room may append any line of text to the log, and there should be some way for other users to stay up-to-date on log additions. We start with a simple implementation, in the style of 20th century Web applications, before it became common to do significant client-side scripting. We evolve toward a version with instant updating upon all message additions, where a chat room runs within a single HTML page updated incrementally by client-side code. Along the way, we highlight our running themes of *encapsulation* and *simple concurrency*.

The examples from this section are written to be understandable to readers with different levels of familiarity with statically typed functional languages. Though the code should be understandable to all at a high level, some remarks (safe to skip) may require more familiarity.

2.1. HTML and SQL

Mainstream modern Web applications manipulate code in many different languages and protocols. Ur/Web hides most of them within a unified programming model, but we decided to expose two languages explicitly: *HTML*, for describing the structure of Web pages as trees, and *SQL*, for accessing a persistent relational database on the server. In contrast to mainstream practice, Ur/Web represents code fragments in these languages as first-class, strongly typed values.

Figure 1 gives our first chat-room implementation, relying on embedding of HTML and SQL code. While, in general, Ur/Web programs contain code that runs on both server and clients, all code from this figure runs on the server, where we are able to enforce that it is run exactly as written in the source code.

The first two lines show declarations of SQL tables, which can be thought of as mutable global variables of type "multiset of records." Table `room`'s records contain integer IDs and string titles, while table `message`'s records contain integer room IDs, timestamps, and string messages. The former table represents the set of available chat rooms, while the latter represents the set of all (timestamped) messages sent to all rooms.

We direct the reader's attention now to the declaration of the `main` function, near the end of Figure 1. Here we see Ur/Web's syntax extensions for embedded SQL and HTML code. Such notation is desugared into calls to constructors of abstract syntax tree types. The `main` definition demonstrates two notations for "antiquoting," or inserting Ur code within a quoted code fragment. The notation `{e}` asks to evaluate expression `e` to produce a subfragment to be inserted at that point, and notation `{[e]}` adds a further stage of formatting `e` as a literal of the embedded language (using type classes[17] as in Haskell's `show`). Note that we are *not* exposing syntax trees to the programmer as strings, so neither antiquoting form presents any danger of *code injection attacks*, where we accidentally interpret user input as code.

What exactly does the `main` definition do? First, we run an SQL query to list all chat rooms. In our tutorial examples, we will call a variety of functions from Ur/Web's standard library, especially various higher-order functions for using SQL query results. Such functions are *higher-order* in the sense that they take other functions as arguments, and we often write those function arguments *anonymously* using the syntax `fn x => e`, which defines a function that, when called, returns the value of expression `e` where parameter variable `x` is replaced with the actual argument value. We adopt a typographic convention for documenting each library function briefly, starting with `queryX1`, used in `main`:

> `queryX1` Run an SQL query that returns columns from a single table (leading to the `1` in the identifier), calling an argument function on every result row. Since just a single table is involved, the input to the argument function is a record with one field per column returned by the query. The argument function should return XML fragments (leading to the `X` in the identifier), and all such fragments are concatenated together, in order, to form the result of `queryX1`.

Ur/Web follows functional languages like Haskell in enforcing *purity*, where expressions may not cause side effects. We allow imperative effects on an "opt-in" basis, with types delineating boundaries between effectful and pure code, following Haskell's technique of monadic IO.[14] For instance, the `main` function here inhabits a distinguished monad for input-output. Thus, we use the `<-` notation to run an effectful computation and bind its result to a variable, and we call the `return` function to lift pure values into trivial

Figure 1. A simple chat-room application.

```
table room : { Id : int, Title : string }
table message : { Room : int, When : time,
                  Text : string }

fun chat id =
 let
  fun say r =
   dml (INSERT INTO message (Room, When, Text)
     VALUES ({[id]}, CURRENT_TIMESTAMP, {[r.Text]}));
   chat id
 in
  title <- oneRowE1 (SELECT (room.Title) FROM room
                     WHERE room.Id = {[id]});
  log <- queryX1 (SELECT message.Text FROM message
                  WHERE message.Room = {[id]}
                  ORDER BY message.When)
                 (fn r => <xml>{[r.Text]}<br/></xml>);
  return <xml><body>
   <h1>Chat Room: {[title]}</h1>

   <form>
    Add message: <textbox{#Text}/>
    <submit value="Add" action={say}/>
   </form>

   <hr/>

   {log}
  </body></xml>
 end

fun main () =
 rooms <- queryX1 (SELECT * FROM room
                   ORDER BY room.Title)
          (fn r => <xml><li><a link={chat r.Id}>
                    {[r.Title]}</a></li></xml>);
 return <xml><body>
  <h1>List of Rooms</h1>

  {rooms}
 </body></xml>
```

computations. Readers unfamiliar with the Haskell style may generally read `<-` as simple variable assignment and `return` in its usual meaning from C-like languages.

The remaining interesting aspect of `main` is in its use of an HTML `<a>` tag to generate a hyperlink. Instead of denoting a link via a URL as in standard HTML, we use a `link` attribute that accepts a *suspended Ur/Web remote function call*. In this case, we call `chat`, which is defined earlier. The Ur/Web implementation handles proper marshalling of arguments in suspended calls.

Now let us examine the implementation of the `chat` function, providing a page for viewing the current message log of a chat room. First, there is a nested definition of a function `say`, which will be called to append a message to the log.

`dml` Run a piece of SQL code for its side effect of mutating the database. The function name refers to SQL's **d**ata **m**anipulation **l**anguage.

This particular invocation of `dml` inserts a new row into the `message` table with the current timestamp, after which we trigger the logic of the main `chat` page to generate output. Note that `say`, like all remotely callable functions, appears to execute *atomically*, so the programmer need not worry about interleavings between concurrent operations by different clients.

The main body of `chat` runs appropriate queries to retrieve the room name and the full, sorted message log.

`oneRowE1` Run an SQL query that should return just one result row containing just a single column (justifying the `1`) that is computed using an arbitrary SQL **e**xpression (justifying the `E`). That one result value becomes the result of the `oneRowE1` call.

We antiquote the query results into the returned page in an unsurprising way. The only new feature involves HTML forms. In general, we tag each input widget with a *record field name*, and then the submit button of the form includes, in its `action` attribute, an Ur/Web function that should be called upon submission, on *a record built by combining the values of all the input widgets*. We will not say any more about HTML forms, which to some extent represent a legacy aspect of HTML that has been superseded by client-side scripting. Old-style forms need to use a rigid language (HTML) for describing how to combine the values of different widgets into one request to send to the server, while these days it is more common to use a Turing-complete language (JavaScript) for the same task.

Compiling an application to run on the real Web platform requires exposing remotely callable functions (like `main`, `chat`, and `say`) via URLs. Ur/Web automatically generates pleasing URL schemes by *serializing the function-call expressions* that appear in places like `link` attributes. For instance, the link to `chat` in the declaration of `main` is compiled to a URL `/chat/NN`, where `NN` is a textual representation of the room ID.

Adding more encapsulation. The application in Figure 1 is rather monolithic. The database state is exposed without restrictions to all parts of the application. We would not tolerate such a lack of encapsulation in a large traditional application. Chunks of functionality should be modularized, for example, into classes implementing data structures. The database tables here are effectively data structures, so why not try to encapsulate them as well?

The answer is that, as far as we are aware, no prior language designs allow it! As we wrote above, the general model is that the SQL database is a preexisting resource, and any part of the application may create an interface to any part of the database. We analogize such a scheme to an object-oriented language where all class fields are public; it forecloses on some very useful styles of modular reasoning. It is important that modules be able to *create their own private database tables*, without requiring any changes to other source code, application configuration files, etc., for the same reason that we do not want client code of a dictionary class to change, when the dictionary switches to being implemented with hash tables instead of search trees.

Figure 2 shows a refactoring of our application code, to present the chat-room table as a mutable abstract data type. We use Ur/Web's module system, which is in the ML tradition.[11] We have *modules* that implement *signatures*, which may impose information hiding by not exposing some members or by making some types *abstract*. Figure 2 defines a module `Room` encapsulating all database access.

The signature of `Room` appears bracketed between keywords `sig` and `end`. We expose an abstract type `id` of chat-room identifiers. Ur/Web code in other program modules will not be able to take advantage of the fact that `id` is really `int`, and thus cannot fabricate new IDs out of thin air. The signature exposes two methods: `rooms`, to list the IDs and titles of all chat rooms; and `chat`, exactly the remotely callable function we wrote before, but typed in terms of the abstract type `id`. Each method's type uses the `transaction` monad, which is like Haskell's IO monad, but with support for executing all side effects *atomically* in a remote call.

The implementation of `Room` is mostly just a copying-and-pasting of the bulk of the code from Figure 1. We only need to add a simple implementation of the `rooms` method.

`queryL1` Return as a list (justifying the `L`) the results of a query that only returns columns of one table (justifying the `1`).

Figure 2. A modular factorization of Figure 1.

```
structure Room : sig
 type id
 val rooms : transaction (list {Id : id,
                                Title : string})
 val chat : id -> transaction page
end = struct
 (* ...copies of old definitions of room, message,
       and chat... *)

 val rooms = queryL1 (SELECT * FROM room
                      ORDER BY room.Title)
end

fun main () =
 rooms <- Room.rooms;
 return <xml><body>
  <h1>List of Rooms</h1>

  {List.mapX (fn r =>
    <xml><li><a link={Room.chat r.Id}>
     {[r.Title]}</a></li></xml>) rooms}
 </body></xml>
```

`List.mapX` Apply an XML-producing function to each element of a list, then concatenate together the resulting XML fragments to compute the result of `mapX`.

The code for `main` changes so that it calls methods of `Room`, instead of inlining database access.

This sort of separation of a data model is often implemented as part of the "model-view-controller" pattern. To our knowledge, that pattern had not previously been combined with guaranteed encapsulation of the associated database tables. It also seems to be novel to apply ML-style type abstraction to database results, as in our use of an `id` type here. We hope this example has helped convey one basic take-away message: giving first-class status to key pieces of Web applications makes it easy to apply standard language-based encapsulation mechanisms.

2.2. Client-side GUI scripting

We will develop two more variations of the chat-room application. Our modifications will be confined to the implementation of the `Room` module. Its signature is already sufficient to enable our experiments, and we will keep the same `main` function code for the rest of the tutorial.

Our first extension takes advantage of *client-side scripting* to make applications more responsive, without the need to load a completely fresh page after every user action. Mainstream Web applications are scripted with JavaScript, but, as in Links and similar languages, Ur/Web scripting is done in the language itself, which is compiled to JavaScript as needed.

Reactive GUIs. Ur/Web GUI programming follows the *functional-reactive* style. That is, to a large extent, the visible page is described as a transformation over *streams* (sequences of values over time). User inputs like keypresses are modeled as primitive streams, which together should be transformed into the stream of page contents that the user sees. Languages like Flapjax[13] and Elm[9] adopt the stream metaphor literally. Ur/Web follows a less pure style, where we retain the *event callbacks* of imperative programming. These callbacks modify *data sources*, which are a special kind of mutable reference cells. The only primitive streams are effectively *the sequences of values that data sources take on*, where new entries are pushed into the streams mostly via imperative code in callbacks. Both flavors of the functional-reactive style provide superior modularity compared to the standard Web model, which involves imperative mutation of the document tree, treated as a public global variable.

As a basic orientation to the concepts of sources and streams in Ur/Web, here is their type signature. We rely on two abstract parameterized types: source α is a data source that can store values of type α, and signal α is a time-varying value that, at any particular time, has some value of type α. As types and values occupy different namespaces, we often reuse an identifier (e.g., source) to stand for both a type and the run-time operation for allocating one of its instances, much as happens with classes and their constructors in Java. We write $\forall\alpha.\ T$ for a type that is *polymorphic* in some arbitrary type parameter α, much as we would write a method prototype like `<a>T f(...)` in Java to bind `a` for use in `T`.

$$
\begin{array}{rcl}
\text{source} & : & \forall\alpha.\ \text{transaction (source } \alpha) \\
\text{get} & : & \forall\alpha.\ \text{source } \alpha \to \text{transaction } \alpha \\
\text{set} & : & \forall\alpha.\ \text{source } \alpha \to \alpha \to \text{transaction } \{\} \\
\\
\text{s\_m} & : & \text{monad signal} \\
\text{signal} & : & \forall\alpha.\ \text{source } \alpha \to \text{signal } \alpha
\end{array}
$$

That is, data sources have operations for allocating them and reading or writing their values. All such operations live inside the transaction monad for imperative side effects. Signals, or time-varying values, happen to form a monad with appropriate operations (e.g., support return and the <- operator), as indicated by the presence of a first-class dictionary s_m witnessing their monadhood. One more key operation with signals is producing them from sources, via the value-level signal function, which turns a source into a stream that documents changes to the source's contents.

Figure 3 demonstrates these constructs in a module implementing a GUI widget for append-only logs. The module signature declares an abstract type `t` of logs. We have methods `create`, to allocate a new log; `append`, to add a new string to the end of a log; and `render`, to produce the HTML representing a log. The type of `render` may be deceptive in its simplicity; Ur/Web HTML values (as in the `xbody` type of HTML that fits in a document body) actually are all implicitly parameterized in the dataflow style, and they are equipped to rerender themselves after changes to the data sources they depend on.

Figure 3. Append-only log module for GUIs.

```
structure Log : sig
 type t
 val create : transaction t
 val append : t -> string -> transaction {}
 val render : t -> xbody
end = struct
 datatype log =
    Nil
  | Cons of string * source log

 type t = {Head : source log,
           Tail : source (source log)}

 val create =
  s <- source Nil;
  s' <- source s;
  return {Head = s, Tail = s'}

 fun append t text =
  s <- source Nil;
  oldTail <- get t.Tail;
  set oldTail (Cons (text, s));
  set t.Tail s;

  log <- get t.Head;
  case log of
    Nil => set t.Head (Cons (text, s))
  | _ => return ()

 fun render_aux log =
  case log of
    Nil => <xml></xml>
  | Cons (text, rest) => <xml>
     {[text]}<br/>
     <dyn signal={log <- signal rest;
                  return (render_aux log)}/>
    </xml>

 fun render t = <xml>
  <dyn signal={log <- signal t.Head;
               return (render_aux log)}/>
 </xml>
end
```

The workhorse data structure of logs is an *algebraic datatype* `log`. Algebraic datatypes are the classic tool for structured data in statically typed functional programming, and they behave similarly to *tagged unions* in some other languages (or Scala's case classes). The definition of `log` gives an exhaustive list of the *constructors* for logs (`Nil` and `Cons`) with the types of arguments that they take, and later we use `case` expressions to deconstruct such values, giving different code to evaluate depending on which constructor built a value. Our definition of `log` looks almost like a standard definition of lists of strings. The difference is that the tail of a nonempty `log` has type `source log`, rather than just `log`. We effectively have a type of lists that supports imperative replacement of list tails, but not replacement of heads.

The type `t` of logs is a record of two fields. Field `Head` is a modifiable reference to the current log state, and `Tail` is a "pointer to a pointer," telling us which source cell we should overwrite next to append to the list. Methods `create` and `append` involve a bit of intricacy to update these fields properly, but we will not dwell on the details. We only mention that the point of this complexity is to avoid rerendering the whole log each time an entry is appended; instead, only a constant amount of work is done per append, to modify the document tree at the end of the log. That sort of pattern is difficult to implement with pure functional-reactive programming.

The most interesting method definition is for `render`. Our prior examples only showed building HTML fragments with no dependencies on data sources. We create dependencies via a pseudotag called `<dyn>`. The `signal` attribute of this tag accepts a signal, a time-varying value telling us what content should be displayed at this point on the page at different times. Crucially, the `signal` monad rules out imperative side effects, instead capturing pure dataflow programming. Since it is a monad, we have access to the usual monad operations `<-` and `return`, in addition to the `signal` function for lifting sources into signals.

Let us first examine the definition of `render_aux`, a recursive helper function for `render`. The type of `render_aux` is `log -> xbody`, displaying a log as HTML. Empty logs are displayed as empty HTML fragments, and nonempty logs are rendered with their textual heads followed by recursive renderings of their tails. (Following functional-programming tradition, we write "head" and "tail" respectively for the first element of a list and the remainder of the list minus that element.) However, the recursive call to `render_aux` is not direct. Instead it appears inside a `<dyn>` pseudotag. We indicate that this subpage depends on the current value of the tail (a data source), giving the computation to translate from the tail's value to HTML. Now it is easy to define `render`, mostly just duplicating the last part of `render_aux`.

An important property of this module definition is that a `rendered` log automatically updates in the browser after every call to `append`, even though we have not coded any explicit coupling between these methods. The Ur/Web runtime system takes care of the details, once we express GUIs via parameterized dataflow.

Client code may use logs without knowing implementation details. In the standard model for programming browser GUIs with JavaScript, mistakes in one code module may wreck subtrees that other modules believe they control. For instance, subtrees are often looked up by string ID, creating the possibility for two different libraries to choose the same ID unwittingly for different subtrees. With the Ur/Web model, the author of module `Log` may think of it as *owning* particular subtrees of the HTML document as private state. Standard module encapsulation protects the underlying data sources from direct modification by other modules, and rendered logs only have dataflow dependencies on the values of those sources.

Remote procedure calls. The GUI widget for displaying the chat log is only one half of the story if we are to write an application that runs within a single page. We also need a way for this application to contact the server, to trigger state modifications and receive updated information. Ur/Web's first solution to that problem is *remote procedure calls* (RPCs), allowing client code to run particular function calls as if they were executing on the server, with access to shared database state. Client code only needs to wrap such calls to remotely callable functions within the `rpc` keyword, and the Ur/Web implementation takes care of all network communication and marshalling. Every RPC appears to execute *atomically*, just as for other kinds of remote calls.

Figure 4 reimplements the `Room` module to take advantage of RPCs and the `Log` widget.

`List.foldl` As in ML, step through a list, applying a function f to each element, so that, given an initial accumulator a and a list $[x_1, \ldots, x_n]$, the result is $f(x_n, \ldots, f(x_1, a)\ldots)$.

`List.app` Apply an effectful function to every element of a list, in order.

The code actually contains few new complexities. Our basic strategy is for each client to maintain the timestamp of the *most recent* chat message it has received. The textbox for user input is associated with a freshly allocated `source string`, via the `<ctextbox>` pseudotag ("c" is for "client-side scripting"). Whenever the user modifies the text shown in this box, the associated source is automatically mutated to contain the latest text. When the user clicks a button to send a message, we run the callback code in the button's `onclick` attribute, *on the client*, whereas the code for this example outside of `on*` attributes runs on the server. This code makes an RPC, telling the server both the new message text and the last timestamp that the client knows about. The server sends back a list of all chat messages newer than that timestamp, and client code iterates over that list, adding each message to the log; and then updates the last-seen timestamp accordingly, taking advantage of the fact that the RPC result list will never be empty, as it always contains at least the message that this client just sent. An `onload` event handler in the `body` tag initialized the log in the first place, appending each entry returned by an initial database query.

Note how seamless the use of the `Log` module is. We allocate a new log, drop its rendering into the right part of the page, and periodically append to it. Pure functional-reactive programming would require some acrobatics to interleave the event streams generated as input to the log system, from the two syntactically distinct calls to `Log.append`.

Figure 4. A client-code-heavy chat-room application.

```
structure Room : sig
 type id
 val rooms : transaction (list {Id : id,
                                Title : string})
 val chat : id -> transaction page
end = struct
 table room : { Id : int, Title : string }
 table message : { Room : int, When : time,
                   Text : string }
 val rooms = queryL1 (SELECT * FROM room
                      ORDER BY room.Title)

 (* New code w.r.t. Figure 2 starts here. *)

 fun chat id =
  let
   fun say text lastSeen =
    dml (INSERT INTO message (Room, When, Text)
         VALUES ({[id]}, CURRENT_TIMESTAMP, {[text]}));
    queryL1 (SELECT message.Text, message.When
             FROM message
             WHERE message.Room = {[id]}
               AND message.When > {[lastSeen]}
             ORDER BY message.When DESC)

   val maxTimestamp =
    List.foldl (fn r acc => max r.When acc) minTime
  in
   title <- oneRowE1 (SELECT (room.Title) FROM room
                      WHERE room.Id = {[id]});
   initial <- queryL1 (SELECT message.Text,
                         message.When
                       FROM message
                       WHERE message.Room = {[id]}
                       ORDER BY message.When DESC);
   text <- source "";
   log <- Log.create;
   lastSeen <- source (maxTimestamp initial);

   return <xml><body onload={
     List.app (fn r => Log.append log r.Text) initial}>
    <h1>Chat Room: {[title]}</h1>

    Add message: <ctextbox source={text}/>
    <button value="Add" onclick={fn _ =>
     txt <- get text;
     set text "";
     lastSn <- get lastSeen;
     newMsgs <- rpc (say txt lastSn);
     set lastSeen (maxTimestamp newMsgs);
     List.app (fn r => Log.append log r.Text)
      newMsgs}/>
    <hr/>
    {Log.render log}
   </body></xml>
  end
end
```

2.3. Message passing from server to client

Web browsers make it natural for clients to contact servers via HTTP requests, but the other communication direction may also be useful. One example is our chat application, where only the server knows when a client has posted a new message, and we would like the server to notify all other clients in the same chat room. Ur/Web presents an abstraction where servers are able to send typed *messages* directly to clients, and the Ur/Web compiler and runtime system implement this abstraction once and for all on top of standard protocols and APIs.

The messaging abstraction is influenced by concurrent programming languages like Erlang[1] and Concurrent ML.[15] Communication happens over unidirectional *channels*. Every channel has an associated client and a type. The server may *send* any value of that type to the channel, which conceptually adds the message to a queue on the client. Clients asynchronously *receive* messages from channels for which they have handles, conceptually dequeuing from local queues, blocking when queues are empty. Any remote call may trigger any number of sends to any number of channels. All sends in a single remote call appear to take place *atomically*.

The API for channels is straightforward:

$$\begin{aligned} \text{channel} &: \forall\alpha.\ \text{transaction}\ (\text{channel}\ \alpha) \\ \text{recv} &: \forall\alpha.\ \text{channel}\ \alpha \to \text{transaction}\ \alpha \\ \text{send} &: \forall\alpha.\ \text{channel}\ \alpha \to \alpha \to \text{transaction}\ \{\} \end{aligned}$$

Figure 5 gives another reimplementation of `Room`, this time using channels to keep clients synchronized at all times, modulo small amounts of lag. We retain the same `room` and `message` tables as before, but we also add a new table `subscriber`, tracking which clients are listening for notifications about which rooms. (Thanks to Ur/Web's approach to encapsulation of database tables, we need not change any other source or configuration files just because we add a new private table.) Every row of `subscriber` has a room ID `Room` and a channel `Chan` that is able to receive strings.

Now the `chat` method begins by allocating a fresh channel with the `channel` operation, which we immediately insert into `subscriber`. Compared to Figure 4, we drop the client-side timestamp tracking. Instead, the server will use channels to notify all clients in a room, each time a new message is posted there. In particular, see the tweaked definition of `say`.

`queryI1` Run an SQL query returning columns from just a single table (justifying the `1`), applying a function to each result in order, solely for its imperative side effects (justifying the `I`).

We loop over all channels associated with the current room, sending the new message to each one.

There is one last change from Figure 4. The `onload` attribute of our `<body>` tag still contains code to run immediately after the page is loaded. This time, before we initialize the `Log` structure, we also create a new thread with the `spawn` primitive. That thread loops forever, blocking to receive messages from the freshly created channel and add them to the log.

Threads follow a simple *cooperative* semantics, where the programming model says that, at any moment in time, at most one thread is running *across all clients of the application*. Execution only switches to another thread when the current one terminates or executes a blocking operation, among which we have RPCs and channel `recv`. Of course, the Ur/Web implementation will run many threads at once, with an arbitrary number on the server and one JavaScript thread per client, but the implementation ensures that no behaviors occur that could not also be simulated with the simpler one-thread-at-a-time model.

This simple approach has pleasant consequences for program modularity. The example of Figure 5 only shows a single program module taking advantage of channels. It is possible for channels to be used freely throughout a program, and the Ur/Web implementation takes care of routing messages to clients, while maintaining the simple thread semantics.

Figure 5 contains no explicit deallocation of clients that have stopped participating. The Ur/Web implementation detects client departure using a heartbeating mechanism.

Figure 5. Final chat-room application.

```
structure Room : sig
 type id
 val rooms : transaction (list {Id : id,
                                Title : string})
 val chat : id -> transaction page
end = struct
 table room : { Id : int, Title : string }
 table message : { Room : int, When : time,
                   Text : string }
 val rooms = queryL1 (SELECT * FROM  room
                      ORDER BY room.Title)

 (* New code w.r.t. Figure 2 starts here. *)

 table subscriber : { Room : int,
                      Chan : channel string }

 fun chat id =
  let
   fun say text =
    dml (INSERT INTO message (Room, When, Text)
         VALUES ({[id]}, CURRENT_TIMESTAMP, {[text]}));
    queryI1 (SELECT subscriber.Chan FROM subscriber
             WHERE  subscriber.Room = {[id]})
             (fn r => send r.Chan text)
  in
   chan <- channel;
   dml (INSERT INTO subscriber (Room, Chan)
        VALUES ({[id]}, {[chan]}));

   title <- oneRowE1 (SELECT (room.Title) FROM room
                      WHERE room.Id = {[id]});
   initial <- queryL1 (SELECT message.Text,
                              message.When
                       FROM message
                       WHERE message.Room = {[id]}
                       ORDER BY message.When DESC);

   text <- source "";
   log <- Log.create;

   return <xml><body onload={
     let
      fun listener () =
       text <- recv chan;
       Log.append log text;
       listener ()
     in
      spawn (listener ());
      List.app (fn r => Log.append log r.Text) initial
     end}>
    <h1>Chat Room: {[title]}</h1>

    Add message: <ctextbox source={text}/>
    <button value="Add" onclick={fn _ =>
     txt <- get text;
     set text "";
     rpc (say txt)}/>

    <hr/>

    {Log.render log}
   </body></xml>
  end
end
```

When a client departs, the runtime system *atomically deletes from the database all references to that client's channels*, providing a kind of modularity similar to what garbage collection provides for heap-allocated objects.

3. RELATED WORK

The space of Web-development tools is a busy one, both in the research world and in the mainstream, so we will not have space to take a nearly complete tour of it; we provide a longer comparison elsewhere.[7]

A variety of research programming languages and libraries were under development in the early 2000s, in that interesting transitional period where the Web had become mainstream but the highly interactive "AJAX" style of Gmail and so on had not yet become common. The PLT Scheme Web Server[10] provides completely first-class support for URLs as first-class functions (more specifically, continuations), making it very easy to construct many useful abstractions. The Links[8] language pioneered strong static checking of multiple-tier Web applications, where the code for all tiers is collected in a single language. Hop[16] is another unified Web programming language, but is dynamically typed and based on Scheme. Ocsigen[2, 3] is an OCaml-based platform for building dynamic Web sites in a unified language, with static typing that rules out many potential programming mistakes. The Opa language[c] is another statically typed unified language for database-backed Web applications.

In broad strokes, how does Ur/Web compare to these others from the languages research community? We believe the key comparison points relate to those features of Ur/Web that we have stressed throughout this article: encapsulation and concurrency. No other languages allow enforced encapsulation of database state inside of modules, and the languages listed above all support client-side GUIs only through mutation of a global document tree, with no enforced encapsulation of client-side state pieces. Their semantics for concurrency are substantially more complicated than Ur/Web's transactions.

Several other languages and frameworks support functional-reactive programming for client-side Web GUIs, including Flapjax,[13] which is available in one flavor as a JavaScript library; and Elm,[9] a new programming language. These libraries implement the original, "pure" version of functional-reactive programming, where key parts of programs are written as pure functions that transform input streams into streams of visible GUI content. Such a style is elegant in many cases, but it does not seem compatible with the modularity patterns we demonstrated in Section 2.2, where it is natural to spread input sources to a single stream across different parts of a program. Ur/Web supports that kind of modularity by adopting a hybrid model, with imperative event callbacks that trigger recomputation of pure code.

As far as we are aware, Ur/Web was the first Web programming tool to support impure functional-reactive programming, but the idea of reactive GUI programming in JavaScript is now mainstream, and too many frameworks exist to detail here.

One popular JavaScript framework is Meteor,[d] distinguished by its support for a particular reactive programming style. It integrates well with mainstream Web development tools and libraries, which is a nontrivial advantage for most programmers. Its standard database support is for MongoDB, with no transactional abstraction or other way of taming simultaneous complex state updates. Like Opa, Meteor allows modules to encapsulate named database elements, but an exception is thrown if two modules

[c] http://opalang.org/.
[d] http://www.meteor.com/.

have chosen the same string name for their elements; module authors must coordinate on how to divide a global namespace. Meteor supports automatic publishing of server-side database changes into client-side caches, and then from those caches into rendered pages. In addition to automatic updating of pages based on state changes, Meteor provides a standard DOM-based API for walking document structure and making changes imperatively, though it is not very idiomatic. Meteor's machinery for reactive page updating involves a more complex API than Ur/Web's. Its central concept is of *imperative* functions that need to be rerun when any of their dependencies change, whereas Ur/Web describes reactive computations in terms of *pure* code within the signal monad, such that it is easy to rerun only *part* of a computation, when not all of its dependencies have changed. Forcing purity on these computations helps avoid the confusing consequences of genuine side effects being repeated on each change to dependencies. The five lines of code near the start of Section 2.2, together with the `<dyn>` pseudotag, give the complete interface for reactive programming in Ur/Web, in contrast with tens of pages of documentation (of dynamically typed functions) for Meteor.

Other popular JavaScript frameworks include Angular.js,[e] Backbone,[f] Ractive,[g] and React.[h] A commonality among these libraries seems to be heavyweight approaches to the basic structure of reactive GUIs, with built-in mandatory concepts of models, views, controllers, templates, components, etc. In contrast, Ur/Web has its 5-line API of sources and signals. These mainstream JavaScript frameworks tend to force elements of reactive state to be enumerated explicitly as fields of some distinguished object, instead of allowing data sources to be allocated dynamically throughout the modules of a program and kept as private state of those modules.

4. CONCLUSION

We have presented the design of Ur/Web, a programming language for Web applications, focusing on a few language-design ideas that apply broadly to a class of distributed applications. Our main mission is to promote two desiderata that programmers should be asking for in their Web frameworks, but which seem almost absent from mainstream discussion: stronger and domain-specific modes of *encapsulation* and *simple concurrency* models based on transactions.

Ur/Web is used in production today for a number of applications, including at least one[i] with thousands of paying customers. We list more examples elsewhere.[7] We have also written elsewhere about Ur/Web's whole-program optimizing compiler,[6] which has placed favorably in a third-party Web-framework benchmarking initiative,[j] for instance achieving the second-best throughput (out of about 150 participating frameworks) of about 300,000 requests per second on the test closest to a full Web app.

[e] https://angularjs.org/.
[f] http://backbonejs.org/.
[g] http://www.ractivejs.org/.
[h] http://facebook.github.io/react/.
[i] http://www.bazqux.com/.
[j] http://www.techempower.com/benchmarks/.

Acknowledgments

This work has been supported in part by National Science Foundation grant CCF-1217501. The author thanks Christian J. Bell, Benjamin Delaware, Xavier Leroy, Clément Pit–Claudel, Benjamin Sherman, and Peng Wang for their feedback on drafts.

References

1. Armstrong, J. Erlang – A survey of the language and its industrial applications. In *Proceedings of the Symposium on Industrial Applications of Prolog (INAP96)* (1996), 16–18.
2. Balat, V. Ocsigen: Typing Web interaction with Objective Caml. In *Proceedings of the 2006 Workshop on ML*, ML '06 (New York, NY, USA, 2006). ACM, 84–94.
3. Balat, V., Vouillon, J., Yakobowski, B. Experience report: Ocsigen, a Web programming framework. In *Proceedings of the 14th ACM SIGPLAN International Conference on Functional Programming*, ICFP '09 (New York, NY, USA, 2009). ACM, 311–316.
4. Cheney, J., Lindley, S., Wadler, P. A practical theory of language-integrated query. In *Proceedings of the 18th ACM SIGPLAN International Conference on Functional Programming*, ICFP '13 (New York, NY, USA, 2013). ACM, 403–416.
5. Chlipala, A. Ur: Statically-typed metaprogramming with type-level record computation. In *Proceedings of the 31st ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '10 (New York, NY, USA, 2010). ACM, 122–133.
6. Chlipala, A. An optimizing compiler for a purely functional web-application language. In *Proceedings of the 20th ACM SIGPLAN International Conference on Functional Programming*, ICFP 2015 (New York, NY, USA, 2015). ACM, 10–21.
7. Chlipala, A. Ur/Web: A simple model for programming the Web. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '15 (New York, NY, USA, 2015). ACM, 153–165.
8. Cooper, E., Lindley, S., Wadler, P., Yallop, J. Links: Web programming without tiers. In *Proceedings of the 5th International Conference on Formal Methods for Components and Objects*, FMCO'06 (Berlin, Heidelberg, 2007). Springer-Verlag, 266–296.
9. Czaplicki, E., Chong, S. Asynchronous functional reactive programming for GUIs. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '13 (New York, NY, USA, 2013). ACM, 411–422.
10. Krishnamurthi, S., Hopkins, P.W., McCarthy, J., Graunke, P.T., Pettyjohn, G., Felleisen, M. Implementation and use of the PLT Scheme Web Server. *Higher Order Symbol. Comput.* 20, 4 (2007), 431–460.
11. MacQueen, D. Modules for standard ML. In *Proceedings of the 1984 ACM Symposium on LISP and Functional Programming*, LFP '84 (New York, NY, USA, 1984). ACM, 198–207.
12. Meijer, E., Beckman, B., Bierman, G. LINQ: Reconciling objects, relations and XML in the .NET framework. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, SIGMOD '06 (New York, NY, USA, 2006). ACM, 706–706.
13. Meyerovich, L.A., Guha, A., Baskin, J., Cooper, G.H., Greenberg, M., Bromfield, A., Krishnamurthi, S. Flapjax: A programming language for Ajax applications. In *Proceedings of the 24th ACM SIGPLAN Conference on Object Oriented Programming Systems Languages and Applications*, OOPSLA '09 (New York, NY, USA, 2009). ACM, 1–20.
14. Peyton Jones, S.L., Wadler, P. Imperative functional programming. In *Proceedings of the 20th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '93 (New York, NY, USA, 1993). ACM, 71–84.
15. Reppy, J.H. *Concurrent Programming in ML*. Cambridge University Press, New York, NY, USA, 1999.
16. Serrano, M., Gallesio, E., Loitsch, F. Hop, a language for programming the Web 2.0. In *Proceedings of the First Dynamic Languages Symposium*, DLS '06 (New York, NY, USA, 2006). ACM.
17. Wadler, P., Blott, S. How to make ad-hoc polymorphism less ad hoc. In *Proceedings of the 16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '89 (New York, NY, USA, 1989). ACM, 60–76.

Adam Chlipala (adamc@csail.mit.edu), MIT CSAIL, Cambridge, MA.

Watch the author discuss his work in this exclusive *Communications* video. http://cacm.acm.org/videos/ur-web

IOWA STATE UNIVERSITY

Lecturer or Senior Lecturer
Electrical and Computer Engineering

The Department of Electrical and Computer Engineering (**http://www.ece.iastate.edu/**) at Iowa State University invites applications for the position of Lecturer or Senior Lecturer at part or full–time. Lecturers serve the department's education mission through instruction, laboratory development, and curriculum innovation. This position is available in the area of software engineering.

Successful candidates will have experience that demonstrates ability, potential, and commitment to undergraduate education.

All faculty members are expected to interact collegially and maintain the highest standards of integrity and ethical behavior. The Department of Electrical and Computer Engineering (College of Engineering) and the Department of Computer Science (College of Liberal Arts and Sciences) jointly administer the Software Engineering Program with a Director overseeing the program.

All interested, qualified persons are encouraged to apply early and must apply for this position by visiting **http://www.iastatejobs.com:80/postings/19005** and complete the Employment Application for vacancy **#600084**. Please be prepared to enter or attach the following: Resume/Curriculum Vitae, letter of application/cover letter, contact information for three references. If you have questions regarding this application process, please email **employment@iastate.edu** or call 515-294-4800 or Toll Free: 1-877-477-7485.

Iowa State University is an Equal Opportunity/Affirmative Action Employer.

Swarthmore College
Computer/Electrical Engineering Faculty (All Ranks)

Swarthmore College invites applications for a tenure-track or tenured position at any rank in the area of Computer/Electrical Engineering, to start during the Fall semester of 2017. A doctorate in Computer or Electrical Engineering or a related field is required. The appointee will pursue a research program that encourages involvement by undergraduate students. Strong interests in undergraduate teaching, supervising senior design projects, and student mentoring are also required. Teaching responsibilities include courses in computer hardware such as computer architecture and digital logic, and electives in the appointee's area of specialization. For the full details on the position and application instructions, visit https://academicjobsonline.org/ajo/jobs/7247.

Swarthmore College has a strong institutional commitment to excellence through diversity in its educational program and employment practices and actively seeks and welcomes applications from candidates with exceptional qualifications, particularly those with demonstrable commitments to a more inclusive society and world.

NJIT
New Jersey Institute of Technology

Dean: College of Computing Sciences

New Jersey Institute of Technology (NJIT) seeks applications and nominations from the academic and corporate sectors for the position of Dean of the Ying Wu College of Computing Sciences (CCS).

The College, established in 2001, is comprised of the Department of Computer Science, the Department of Information Systems, and the Information Technology Program. The College enrolls approximately 2385 students, including 1390 undergraduates, 925 master students, and 70 doctoral students; these students, representing about 21 percent of NJIT's student body, are taught by 34 tenured and tenure-track faculty and 20 full-time, non-tenure track faculty. Undergraduate degree programs in Computer Science, Information Systems, and Information Technology are ABET accredited. CCS's current research strengths include, but are not limited to, Cyber Security and Networking, Data Analytics, Cyber-Human Systems, and Game Design. Based on the 2015 Shanghai Rankings, TheBestSchools.org released its top 100 Computer Science programs in the world, with NJIT's program ranked 47 in the United States and 100 globally. The Princeton Review chose NJIT as one of the top 50 undergraduate schools for Game Design in 2016.

Responsibilities: The Dean serves as the chief executive officer of the College with primary responsibility for enhancing its national leadership in computing research and education. The Dean is a senior academic officer who reports to the Provost and works closely with the President, the deans of the other five colleges and senior officers to advance the university's mission. Specific responsibilities include strategic planning, corporate outreach, fundraising leadership to attract resources necessary to sustain and enhance the College, program evaluation and development, and recruitment of new faculty. The Dean manages the college's finances and budget and leads CCS's marketing and visibility initiatives. The Dean is also expected to foster interdisciplinary education and research programs as a priority of the University's 2020 Vision Strategic Plan.

Qualifications: Candidates must possess an earned doctorate and a distinguished record of teaching, research, publication, funding, and service that merits appointment as a full professor with tenure in CCS. We seek an individual who can span disciplinary boundaries to build alliances and partnerships for CCS not only with other colleges, but also with private sector organizations and government agencies. The successful candidate will be a dynamic leader with a demonstrated ability to promote successful research, service, teaching and learning outcomes and offer innovative solutions to the challenges facing higher education today and into the future.

Application: To apply, please visit https://njit.jobs and search for posting # 603260.

Deadline for applications to be submitted in order to guarantee search committee's full consideration is August 20, 2016. Please address any questions regarding this search to caudill@njit.edu or gwang@njit.edu.

To build a diverse workforce, NJIT encourages applications from individuals with disabilities, minorities, veterans and women. EEO employer.

NEW JERSEY INSTITUTE OF TECHNOLOGY • UNIVERSITY HEIGHTS, NEWARK, NJ 07102-1982

[CONTINUED FROM P. 104] fingers, but it's just the frame and the forks and the mudguards. The saddles look weird before they're fully grown, on the trunks like cankers, but that's all they are. Growths, like the seats we're sitting on. Don't get up. It'll only go out of shape.

There's a skill to the job, you know. Say you see a growth that's going wild, or a stray sapling from another plantation, and you have to prune it before it spreads. OK, I'll put the knife away. I don't want you to feel uncomfortable.

Settle in. Take it easy.

I know, I know. The lab boys—and girls of course—will tell you it's all predictable, like a chemical factory. But biology's not like that, even when it's synthetic. I'm not mystical about it. It's just mutation and natural selection. We have a joke—that we're for intelligent design, and against evolution. Intelligent design is what the company does, and evolution is what it pays us to prevent.

But you can't. Not completely.

Time for another. No, not at all, my treat. You're our guest here. And have a shot of spirit on the side; yes, we do our own distilling. Careful though, you're probably not used to it.

Just a sip, that's it. Oh! There's a napkin. A gulp of beer afterward, that's the trick. *Gesundheit!* Here, have another napkin.

Where were we? Oh yes. This tax business you're on about. Well, the company takes care of all that. We get our bank statements on our phones. Take a look. Yes, the balance does mount up. Not much we need to spend money on here. Every so often some government department sends in someone young and keen like yourself who thinks they can make a big deal of some little thing and make a name for themselves. But they soon come round, every one of them.

Take Jess over there. Lovely girl. No, she's not smoking. Good grief, do you think we're crazy? It's some kind of steam, you just roll up the leaf real tight like, and it heats up in the center. Some experimental thing that blew in, no doubt. Like I was saying, evolution.

Got him right there, admitting a breach of the GM regulations. My cheerful host waves a hand around the crowded recreation shelter, oblivious to his own carelessness ... but the offense is trivial compared to the tax evasion, and I doubt he knows anything about how the company pulls it off.

Anyway, Jess. Oh, hi. Just telling him about you, but I'm sure you can ... ah, be off with you!

That woman will be the death of me some day. Likes her fun. Loves her work. But you should have seen her when she came here two years ago. Health and Safety inspector. Hard hat and high heels and a clipboard. Wouldn't touch the beer, insisted on fruit juice. None of ours, either. She took a bottle from her own handbag and sipped from it.

Sit yourself down, I said, and got a beer for myself and started talking to her, just like I'm talking to you. After a while she began to come round to my way of seeing things, just like you are.

So that's *what became of the young lady from Environment ... I find myself enjoying the sound of the forester's voice, and somehow that lazy enjoyment is more important than the discovery. Must be the vague benign glow from the drink. The drink! Have I been slipped something?*

Nah, don't look at your pint like that, or even your shot. It's good beer and good grain spirit with a touch of flavoring, that's all. It's not what's bringing you round. And it's not what's been making you sneeze.

It's the spores.

Told you, the seat's made from the same stuff as the bike saddles. Just another mutant growth. But it's not just the shape that's mutated.

The spores waft out whenever you shift your weight. You can't help but inhale them. The people who designed the bikes wanted people to feel confident riding them, so they did a bit of tinkering with the spores. Put a bit of genetic code in to work on the old nervous system. Give feelings of confidence and rightness and wellbeing.

So that's *why LifeCycles bikes sell so well, why their customers are so loyal. And their employees, too ... and why everyone who finds out about this massive, brazen violation of the GM regulations and all the relevant international conventions keeps quiet about it. And why every tax inspector who goes in finds the accounts add up. Or if they don't, they like the people here too much to tell on them.*

Just as I won't.

The funny thing is, I should be feeling outraged, but I'm not. I'm laughing at the ingenuity and effrontery of it all. I look at the guy, grinning stupidly. There's something I want to tell him. It's just on the tip of my tongue. But I can't.

And our bar seats have the same effect on anyone who sits on them—good feelings.

Gut feelings, you might say. Because that's where they come from—your intestinal flora tweaked by the spores to pump out mood-modifying molecules to your bloodstream.

Doesn't mean they're not real feelings.

You'll like it here.

And I did, for several days. The feelings were real all right. I loved the place. I looked around and considered what job to apply for, and kept phoning in glowing reports to the office. Meanwhile, the polycarbonate strands under my skin kept reporting back what was really going on. Then, the next morning, I woke to the sound of helicopters.

The Revenue knew what the wire had reported, but back at the office they got nothing out of me. Geraldine was very understanding, and put me on sick leave until the artificial loyalty wore off.

Now it has, I can tell the story. But one question still bothers me: What stopped me, for those days of synthetic love for the company, from telling them about the wire?

Only now, when the wire has disintegrated, do I realize one obvious answer—the wire itself. The Revenue has its own biochemical tricks to ensure loyalty and secrecy. And it isn't telling anyone about them.

When I ask Geraldine if I'm right, she looks vague.

'It's just on the tip of my tongue,' she says. 'It'll come back to me.' C

Ken MacLeod (kenneth.m.macleod@gmail.com) is the author of 15 novels, from *The Star Fraction* (Orbit Books, London, 1995) to *The Corporation Wars: Dissidence* (Orbit Books, London, 2016). He blogs at The Early Days of a Better Nation (http://kenmacleod.blogspot.com) and tweets as @amendlocke.

From the intersection of computational science and technological speculation, with boundaries limited only by our ability to imagine what could be.

DOI:10.1145/2962577

Ken MacLeod

Future Tense
Gut Feelings

Even a little genetic engineering can render us too comfortable for our own good.

IMAGE BY ANDRIJ BORYS ASSOCIATES/SHUTTERSTOCK

I'M A TAX INSPECTOR. *I don't expect to be liked.*

'They're up to something in there,' said Geraldine Myles, her finger-tap casting a black shadow on the map projection. 'I can feel it.'

'We can all feel it, chief,' I said. 'It's getting the evidence that's the problem.'

The LifeCycles forest covered most of Fife. Its export record didn't square with its revenue. The product's popularity was as inarguable as it was unaccountable. Every inspector we'd sent in had returned glowing reports. The young lady from Environment was so impressed she'd chucked in her job and joined it. Drones? The forest found a way to swat them. The tiny wreckage was always returned, with a bland apology and a note on commercial secrecy.

'Evidence? That's where you come in.' Myles shot me an evil smile. 'Or rather, that's where you're going *in.'*

Talk in private? Sure, happy to do that. We've got nothing to hide. Everything's open and above board, even if it's a bit dark and gloomy looking. But what's private these days? The trees don't talk, but the walls listen; that's what we say around here.

Walls? I'm confused for a moment, until I realize this friendly foreman is referring to the densely interwoven genetically modified hedge that surrounds and covers this place deep in the forest. It's one of the LifeCycles recreation spots, modeled on an old-fashioned pub. The lighting is bioluminescent tubes. The floor is like pine needles, the furnishings cork. The bar at the center is conventional polished hardwood, but seamless in a way that makes me suspect it's GM, too. A couple dozen people are here, at the end of the working day. Folks come and go through gaps in the hedge. Talk is careless when loud. I can't hear what it's like when it's quiet. But the wire can.

It's not the walls he should be worrying about.

We call the transmitter injected across my collarbones 'the wire,' but that's just a skeuomorph. It's a new device. It'll be about a fortnight before it disintegrates. Subcutaneous, organic, undetectable ... and now, with a carefully casual shrug of my shoulders ...

On.

Sit yourself down and let me get you a pint. No, no ... it's nothing, it's a local brew, you can see the kegs right there. Yes, it's all covered in the forest bylaws; you can look it up.

Cheers! Good stuff, eh?

You were saying? Oh yes! The trees. Well, I can see how you might get that impression. A lot of folks do, the first time. Oppressive and sinister, they tell me. It's dark beneath the trees—but the leaves are designed to catch as much sunlight as possible.

And the branches ... nothing but bicycle frames, thousands of them. It looks sinister only when you let daft ideas into your head, about a crooked arm, maybe, with clawed [CONTINUED ON P. 103]

SUI 2016 TOKYO

ACM Symposium on
Spatial User Interaction
Tokyo, Japan
15-16 October 2016
Co-located with ACM UIST

Sponsored by

Association for Computing Machinery
SIGCHI
ACM SIGGRAPH
SigMR

http://sui-symposium.org

Welcome to ***SUI 2016*** (rhymes with GUI, スゥイ), is a new and upcoming venue for the dissemination of future user interfaces. We invite you to enjoy presentations of leading-edge research results in a comfortable atmosphere. ***Selected SUI papers and demos will be presented in a special poster/ demo track at UIST 2016.***

Deadlines
Paper Submissions: 1 July 2016
Poster and Demo Submissions: 9 August 2016

Panelists:
Otmar Hilliges
(ETH Zürich)
Hrvoje Benko
(Microsoft Research)
Alex Olwal
(Google Inc.)

Moderator:
Aitor Rovira
(Nara Institute of
Science and Technology)