

COMMUNICATIONS

CACM.ACM.ORG

OF THE

ACM

02/2013 VOL.56 NO.2

The Tail at Scale

Association for
Computing Machinery



allrecipes 



Cook up the next great app.

Opportunity doesn't just knock. In the Windows Store it swipes, taps and clicks, too. See how Allrecipes and others are building immersive apps for the new Windows experience and learn how you can put your app in the hands of new users everywhere.

Build for the new Windows Store.
Open for business at windowsstore.com

 **Windows 8**

SENSE THE TRANSFORMATION



SIGGRAPH
ASIA2013
HONG KONG

CONFERENCE 19 NOV - 22 NOV
EXHIBITION 20 NOV - 22 NOV

HONG KONG CONVENTION
AND EXHIBITION CENTRE

SA2013.SIGGRAPH.ORG

LEAD SPONSOR



SPONSORED BY



Departments

- 5 **Letter from ACM Publications Board Co-Chairs**
Positioning ACM for an Open Access Future
By Ronald F. Boisvert and Jack W. Davidson
-
- 7 **From the President**
Growing the ACM Family
By Vinton G. Cerf
-
- 8 **BLOG@CACM**
When Reviews Do More than Sting
Bertrand Meyer wonders why malicious reviews run rampant in computer science.
-
- 35 **Calendar**
-
- 103 **Careers**

Last Byte

- 112 **Puzzled**
Tumbling Dice
By Peter Winkler

News



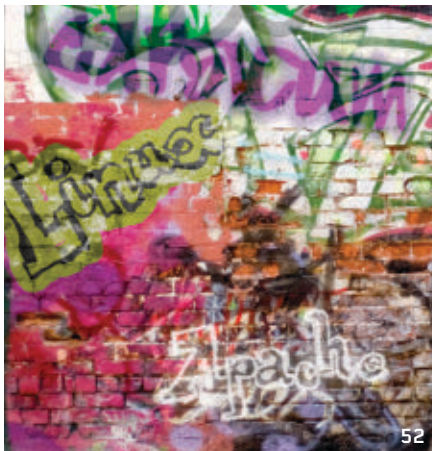
- 11 **Life in Simulation**
Computational models are tackling the complexity of biology, from single-celled microbes to human organs.
By Neil Savage
-
- 14 **Rewing the Rover**
The new Mars rover has attracted plenty of attention for its planetary gymnastics, but the big breakthroughs are under the hood.
By Alex Wright
-
- 17 **A New Model for Healthcare**
Computer modeling is radically redefining healthcare and epidemiology by providing new tools for understanding the impact of different intervention strategies.
By Samuel Greengard

Viewpoints

- 20 **Privacy and Security**
The Tangled Web We Have Woven
Seeking to protect the fundamental privacy of network interactions.
By Eben Moglen
-
- 23 **Inside Risks**
More Sight on Foresight
Reflecting on elections, natural disasters, and the future.
By Peter G. Neumann
-
- 26 **Kode Vicious**
Divided by Division
Is there a “best used by” date for software?
By George V. Neville-Neil
-
- 28 **Education**
Reflections on Stanford’s MOOCs
New possibilities in online education create new challenges.
By Steve Cooper and Mehran Sahami
-
- 31 **Economic and Business Dimensions**
The Value of Microprocessor Designs
Applying a centuries-old technique to modern cost estimation.
By Ana Aizcorbe, Samuel Kortum, and Unni Pillai
-
- 33 **Viewpoint**
Cloud Services Certification
How to address the lack of transparency, trust, and acceptance in cloud services.
By Ali Sunyaev and Stephan Schneider
-
- 37 **Viewpoint**
The Explosive Growth of Postdocs in Computer Science
Considering the factors influencing the recent rapid increase in the number of postdoctoral positions in computer science.
By Anita Jones



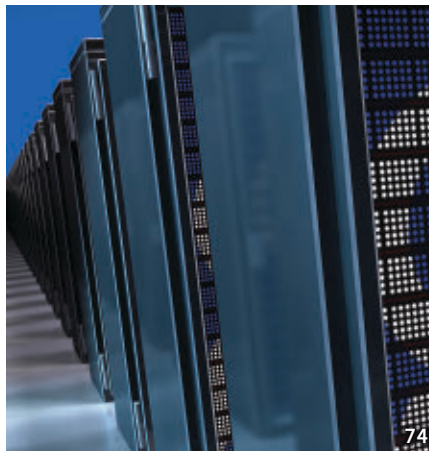
Practice



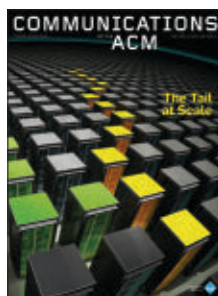
- 40 **Rethinking Passwords**
Our authentication system is lacking. Is improvement possible?
By William Cheswick
-
- 45 **Thinking Methodically about Performance**
The USE method addresses shortcomings in other commonly used methodologies.
By Brendan Gregg
-
- 52 **A Decade of OS Access-Control Extensibility**
Open source security foundations for mobile and embedded devices.
By Robert N.M. Watson

Q Articles' development led by acmqueue.queue.acm.org

Contributed Articles

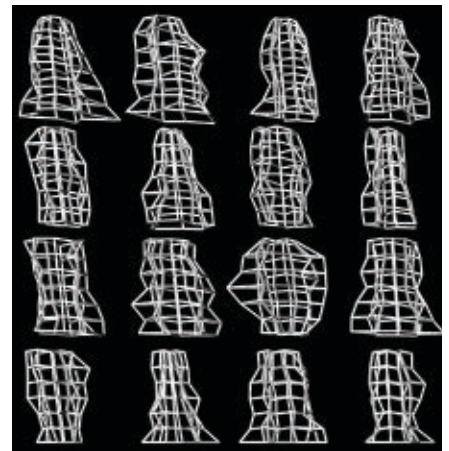


- 64 **New Approaches to Security and Availability for Cloud Data**
Extending the data trust perimeter from the enterprise to the public cloud requires more than encryption.
By Ari Juels and Alina Oprea
-
- 74 **The Tail at Scale**
Software techniques that tolerate latency variability are vital to building responsive large-scale Web services.
By Jeffrey Dean and Luiz André Barroso

**About the Cover:**

The technologies needed to shave milliseconds off the time systems respond to user actions are critical for enhancing the fluid nature of the query action. This month's cover story (p. 74) explores the power of tail-tolerant techniques that allow for higher system utilization without sacrificing responsiveness. Cover illustration by Giacomo Marchesi.

Review Articles



- 82 **Symbolic Execution for Software Testing: Three Decades Later**
The challenges—and great promise—of modern symbolic execution techniques, and the tools to help implement them.
By Cristian Cadar and Koushik Sen

Research Highlights

- 92 **Technical Perspective Is Dark Silicon Real?**
By Pradip Bose
-
- 93 **Power Challenges May End the Multicore Era**
By Hadi Esmaeilzadeh, Emily Blem, Renée St. Amant, Karthikeyan Sankaralingam, and Doug Burger



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.

Executive Director and CEO

John White
Deputy Executive Director and COO
 Patricia Ryan
Director, Office of Information Systems
 Wayne Graves
Director, Office of Financial Services
 Russell Harris
Director, Office of SIG Services
 Donna Cappel
Director, Office of Publications
 Bernard Rous
Director, Office of Group Publishing
 Scott E. Delman

ACM COUNCIL

President
 Vinton G. Cerf
Vice-President
 Alexander L. Wolf
Secretary/Treasurer
 Vicki L. Hanson
Past President
 Alain Chesnais
Chair, SGB Board
 Erik Altman
Co-Chairs, Publications Board
 Ronald Boisvert and Jack Davidson
Members-at-Large
 Eric Allman; Ricardo Baeza-Yates;
 Radia Perlman; Mary Lou Soffa;
 Eugene Spafford
SGB Council Representatives
 Brent Hailpern; Joseph Konstan;
 Andrew Sears

BOARD CHAIRS

Education Board
 Andrew McGettrick
Practitioners Board
 Stephen Bourne

REGIONAL COUNCIL CHAIRS

ACM Europe Council
 Fabrizio Gagliardi
ACM India Council
 Anand S. Deshpande, PJ Narayanan
ACM China Council
 Jianguang Sun

PUBLICATIONS BOARD

Co-Chairs
 Ronald F. Boisvert; Jack Davidson
Board Members
 Marie-Paule Cani; Nikil Dutt; Carol Hutchins;
 Joseph A. Konstan; Ee-Peng Lim;
 Catherine McGeoch; M. Tamer Ozsu;
 Vincent Shen; Mary Lou Soffa

ACM U.S. Public Policy Office

Cameron Wilson, Director
 1828 L Street, N.W., Suite 800
 Washington, DC 20036 USA
 T (202) 659-9711; F (202) 667-1066

Computer Science Teachers Association

Chris Stephenson,
 Executive Director

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.

STAFF

DIRECTOR OF GROUP PUBLISHING

Scott E. Delman
 publisher@cacm.acm.org

Executive Editor

Diane Crawford

Managing Editor

Thomas E. Lambert

Senior Editor

Andrew Rosenbloom

Senior Editor/News

Jack Rosenberger

Web Editor

David Roman

Editorial Assistant

Zarina Strakhan

Rights and Permissions

Deborah Cotton

Art Director

Andrij Borys

Associate Art Director

Margaret Gray

Assistant Art Directors

Mia Angelica Balaquiot

Brian Greenberg

Production Manager

Lynn D'Addesio

Director of Media Sales

Jennifer Ruzicka

Public Relations Coordinator

Virginia Gold

Publications Assistant

Emily Williams

Columnists

Alok Aggarwal; Phillip G. Armour;
 Martin Campbell-Kelly;
 Michael Cusumano; Peter J. Denning;
 Shane Greenstein; Mark Guzdial;
 Peter Harsha; Leah Hoffmann;
 Mari Sako; Pamela Samuelson;
 Gene Spafford; Cameron Wilson

CONTACT POINTS

Copyright permission
 permissions@cacm.acm.org

Calendar items
 calendar@cacm.acm.org

Change of address
 acmhlp@acm.org

Letters to the Editor
 letters@cacm.acm.org

WEBSITE

http://cacm.acm.org

AUTHOR GUIDELINES

http://cacm.acm.org/guidelines

ACM ADVERTISING DEPARTMENT

2 Penn Plaza, Suite 701, New York, NY
 10121-0701
 T (212) 626-0686
 F (212) 869-0481

Director of Media Sales

Jennifer Ruzicka
 jen.ruzicka@hq.acm.org

Media Kit acmm mediasales@acm.org

Association for Computing Machinery (ACM)

2 Penn Plaza, Suite 701
 New York, NY 10121-0701 USA
 T (212) 869-7440; F (212) 869-0481

EDITORIAL BOARD

EDITOR-IN-CHIEF

Moshe Y. Vardi
 eic@cacm.acm.org

NEWS

Co-Chairs

Marc Najork and Prabhakar Raghavan

Board Members

Hsiao-Wuen Hon; Mei Kobayashi;
 William Pulleyblank; Rajeev Rastogi

VIEWPOINTS

Co-Chairs

Susanne E. Hambrusch; John Leslie King;
 J Strother Moore

Board Members

William Aspray; Stefan Bechtold; Judith
 Bishop; Stuart I. Feldman;
 Peter Freeman; Seymour Goodman;
 Mark Guzdial; Richard Heeks;
 Rachele Hollander; Richard Ladner;
 Susan Landau; Carlos Jose Pereira de Lucena;
 Beng Chin Ooi; Loren Terveen;
 Jeannette Wing

PRACTICE

Chair

Stephen Bourne

Board Members

Eric Allman; Charles Beeler; Bryan Cantrill;
 Terry Coatta; Stuart Feldman; Benjamin Fried;
 Pat Hanrahan; Tom Limoncelli;
 Marshall Kirk McKusick; Erik Meijer;
 George Neville-Neil; Theo Schlossnagle;
 Jim Waldo

The Practice section of the CACM

Editorial Board also serves as
 the Editorial Board of *COMMUNIQUE*.

CONTRIBUTED ARTICLES

Co-Chairs

Al Aho and Georg Gottlob

Board Members

William Aiello; Robert Austin; Elisa Bertino;
 Gilles Brassard; Kim Bruce; Alan Bundy;
 Peter Buneman; Erran Carmel;
 Andrew Chien; Peter Druschel; Carlo Ghezzi;
 Carl Gutwin; James Larus; Igor Markov;
 Gail C. Murphy; Shree Nayar; Bernhard
 Nebel; Lionel M. Ni; Sriram Rajamani;
 Marie-Christine Rousset; Avi Rubin;
 Krishan Sabnani; Fred B. Schneider;
 Abigail Sellen; Ron Shamir; Yoav Shoham;
 Marc Snir; Larry Snyder; Manuela Veloso;
 Michael Vitale; Wolfgang Wahlster;
 Hannes Werthner; Andy Chi-Chih Yao

RESEARCH HIGHLIGHTS

Co-Chairs

Stuart J. Russell and Gregory Morrisett

Board Members

Martin Abadi; Sanjeev Arora; Dan Boneh;
 Andrei Broder; Stuart K. Card; Jon Crowcroft;
 Alon Halevy; Monika Henzinger;
 Maurice Herlihy; Norm Jouppi;
 Andrew B. Kahng; Xavier Leroy;
 Mendel Rosenblum; Ronit Rubinfeld;
 David Salesin; Guy Steele, Jr.; David Wagner;
 Alexander L. Wolf; Margaret H. Wright

WEB

Chair

James Landay

Board Members

Gene Golovchinsky; Marti Hearst;
 Jason I. Hong; Jeff Johnson; Wendy E. MacKay



ACM Copyright Notice

Copyright © 2013 by Association for Computing Machinery, Inc. (ACM). Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and full citation on the first page. Copyright for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or fee. Request permission to publish from permissions@acm.org or fax (212) 869-0481.

For other copying of articles that carry a code at the bottom of the first or last page or screen display, copying is permitted provided that the per-copy fee indicated in the code is paid through the Copyright Clearance Center; www.copyright.com.

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 \$100.

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 acmhlp@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



Association for Computing Machinery



Printed in the U.S.A.

Positioning ACM for an Open Access Future

THE AGE OF open access is upon us. Increasingly, the consensus of authors of research articles and their funding institutions is that the fruits of taxpayer-supported research should be freely available to the public. This is a compelling argument and a noble goal.

But, achieving open access is not easy. Professional maintenance and distribution of large digital archives, guaranteed for the long term, does incur significant cost. The most promising model for recovering such costs under an open-access regime is an author-pays (or, in effect, a funding institution pays) model. Such a scheme introduces issues of its own. If publishers generate revenue by producing *more* content (paid for by authors) rather than *quality* content (paid for by subscribers), then the natural tendency in the system will be for the generation of large quantities of low-quality content. Indeed, we have seen the rise of predatory publishers, actively seeking authors to pay for publication in venues devoid of the exacting scrutiny of conscientious peer review. The result is a glut of third-rate publications that add noise rather than insight to the scientific enterprise.

The important question is: Can we establish a sustainable economic model for publication that serves the interest of both authors and the reading public? We submit that non-profit professional societies must play a critical role in this regard. They are the hallmark of quality in publications, and must remain so to serve the interests of the reading public. But, how do we transition from the current subscription model to a new financial model enabling open access in a way that does not bankrupt the organization in the process? This question has occupied the attention of the ACM Publications Board for several years. Because the stakes are high, the Board has chosen to move with caution.

Because we do not have a reliable crystal ball, we have chosen to provide an array of options for ACM authors and Special Interest Groups (SIGs) to enable a natural, slow, and (hopefully) stable evolution of the publication en-

terprise into the future. Examples of this are ACM's long-standing policies enabling author-produced versions of ACM-published materials to be posted on author Web pages, on the pages of their institutions, and on archives mandated by funding institutions. A shining example of such green open-access policies is ACM's Author-Izer service, which allows authors to place specialized links on their Web pages that tunnel through ACM's paywall to provide free access to the definitive versions of their papers, while capturing download statistics displayed in the ACM Digital Library (DL).

In the next few months ACM will roll out more options for authors and SIGs, which will provide even greater levels of flexibility with regard to open access.

1. *Author-pays Open Access Option.* Individual authors (or their institutions) will have the option to pay a fee that covers ACM publication and curation costs at the time of publication, after which the article will be freely available via the ACM DL platform. The fee, which has yet to be set, will differentiate between journal articles and conference papers, as well as between ACM members and non-members.

2. *Open Access During the Period Around SIG Conferences Option.* SIGs will have the option to make the proceedings from their conferences freely available via the ACM DL platform for up to two weeks before the event and up to two weeks after. Not only will this option facilitate easy access to the proceedings by conference attendees, it will also enable the community at large to experience the excitement of learning about the latest developments being presented in the period surrounding the event itself.

3. *Open Access for Most Recent Instance of SIG Conferences Option.* SIGs will have the option to maintain tables-of-contents for the most recent instance of its conferences on the conference website with ACM Author-Izer links that provide free access to the definitive version of the article maintained in the ACM DL. For conferences that are not in a recurring series, such

access will be enabled for up to one year. When selected by the sponsoring SIG, this reverse embargo will provide open access to conference papers during the period in which they are of greatest interest.

4. *Options for Rights Management.* When publishing articles with ACM, authors will have three options for the management of publication rights for their work. Authors who desire the convenience of having ACM maintain rights and permissions associated with their works will continue to be able to execute copyright transfer to ACM. ACM's copyright policy will continue to grant authors liberal rights for the reuse of their work and the posting of personal versions. Authors who prefer to retain copyright or want a more explicit publishing contract will have the option of signing one of two license agreements. The first will have terms similar to that of the existing copyright transfer. However, this option will provide increased clarity about the rights of both parties. Finally, those authors who wish to retain all rights to their work can do so by exercising the author-pays open access option described previously. In that case all rights remain with the author and ACM is simply granted a permanent license to distribute.

The set of changes unveiled here are but another step in an ongoing process in which ACM adapts to the new realities of scholarly publishing. Many further refinements are possible. For example, if the author-pays option is successful, the Publications Board may consider the feasibility of making previously published papers in the DL open access, or making entire proceedings open access using funding from registration fees or sponsors. As the Board considers each change it will continue to be responsive to the needs and wishes of the ACM community while fulfilling its responsibility to maintain a healthy and sustainable publications program. ■

Ronald F. Boisvert and Jack W. Davidson are co-chairs of the ACM Publications Board.

© 2013 ACM 0001-0782/13/02



Association for
Computing Machinery

Advancing Computing as a Science & Profession

membership application & digital library order form

Priority Code: AD13

You can join ACM in several easy ways:

Online
<http://www.acm.org/join>

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

Fax
+1-212-944-1318

Or, complete this application and return with payment via postal mail

Special rates for residents of developing countries:
<http://www.acm.org/membership/L2-3/>

Special rates for members of sister societies:
<http://www.acm.org/membership/dues.html>

Please print clearly

Name _____

Address _____

City _____ State/Province _____ Postal code/Zip _____

Country _____ E-mail address _____

Area code & Daytime phone _____ Fax _____ Member number, if applicable _____

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

I agree with the Purposes of ACM:

Signature _____

ACM Code of Ethics:
<http://www.acm.org/about/code-of-ethics>

choose one membership option:

PROFESSIONAL MEMBERSHIP:

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

STUDENT MEMBERSHIP:

- ACM Student Membership: \$19 USD
- ACM Student Membership plus the ACM Digital Library: \$42 USD
- ACM Student Membership PLUS Print CACM Magazine: \$42 USD
- ACM Student Membership w/Digital Library PLUS Print CACM Magazine: \$62 USD

All new ACM members will receive an
ACM membership card.
For more information, please visit us at www.acm.org

Professional membership dues include \$40 toward a subscription to *Communications of the ACM*. Student membership dues include \$15 toward a subscription to *XRDS*. Member dues, subscriptions, and optional contributions are tax-deductible under certain circumstances. Please consult with your tax advisor.

RETURN COMPLETED APPLICATION TO:

Association for Computing Machinery, Inc.
General Post Office
P.O. Box 30777
New York, NY 10087-0777

Questions? E-mail us at acmhelp@acm.org
Or call +1-800-342-6626 to speak to a live representative

Satisfaction Guaranteed!

payment:

Payment must accompany application. If paying by check or money order, make payable to ACM, Inc. in US dollars or foreign currency at current exchange rate.

- Visa/MasterCard American Express Check/money order
- Professional Member Dues (\$99 or \$198) \$ _____
- ACM Digital Library (\$99) \$ _____
- Student Member Dues (\$19, \$42, or \$62) \$ _____
- Total Amount Due** \$ _____

Card # _____

Expiration date _____

Signature _____



Vinton G. Cerf

DOI:10.1145/2408776.2408778

Growing the ACM Family

I have been thinking about the demographics of the computing profession and wondering what steps ACM and its members might take to increase interest in this career across a full

spectrum of potential candidates. I spent a good part of a day browsing around in the ACM website discovering that this topic is and has been on the table in many different venues. One obvious place to look was in ACM publications^a and I found many that had stories on this topic: *Communications*, *ACM Inroads*, *ACM Queue*, *eLearn*, *XRDS* (formerly *Crossroads*, the student magazine), *Ubiquity*, *interactions*, among others. I also checked for blogs and found a bunch^b including *Communications*, *ACM Inroads*, *ACM Queue*, *eLearn*, *USACM*, *ACM-W* and *CS-TA-advocate*. Then I looked at the Special Interest Groups^c and found many that seemed likely to be addressing this topic: Computer Science Education (SIGCSE), Computers and Society (SIGCAS), Access (SIGACCESS), Information Technology Education (SIGITE), University and College Computing Services (SIGUCC). It would not surprise me to find I have missed some key publications or that other SIGs that tend to be more technically focused have also, at least on occasion, addressed this same question.

Then I found the Educational Activities page^d that had even more information and links to publications, organizations, and activities. And there is, of course, a wide range of activities focused on engaging women in computing led by *ACM-W*, the ACM Women's Council.^e

If you accept that we do not have nearly as wide a range of participants as desired in the computing profession, the question is whether we can take additional steps to foster interest in this field. There are all kinds of extramural activities that draw young people into computing. For example, the FIRST robotics competitions^f and CAMPUS PARTY^g and the ACM International Collegiate Programming Contest (ACM-ICPC^h). There are many more such activities, several of them focused on computer and network security.

On top of all that, we have the emerging Massive Open Online Courses (MOOCs) phenomenon that may involve hundreds of thousands of participants in all age groups. I am not even going to try to give you a list of those—just employ your favorite search engine and you should reap a long list of Web pages, reports, news articles, and other references to this new use of the Internet.

Indeed, there is no dearth of effort on the educational and activities side to stimulate interest in all aspects of computing but there persists a sense that the demographics of computing are still skewed in many ways. One possibility is that the statistics are wrong and we are not measuring the full range of participation. What about people who develop applications for mobile devices? What about Web page designers? Perhaps we might not be

comfortable including many of these participants in the definition of computing professional—is that an issue? On the other hand, the statistics may be telling us that despite our varied efforts, we are not awakening interest in the field broadly enough. Considering that this field involves primarily thinking logically, designing, implementing, testing of software and hardware, it is difficult to imagine that in and of itself, the profession has any built-in biases against anyone. Of course, not everyone is interested in this kind of activity but anyone who is interested should not be excluded by virtue of any inherent constraints.

As the twenty-first century continues to unfold, we are surrounded by a growing number of devices with programmable features. Software (and hardware) is everywhere. Even casual users need to know something about the nature of this topic. Of course, many people use the products of computing without much musing about its origins: computer games, laptops, desktops, tablets and mobiles to say nothing of cloud computing are all part of daily life, even when their complexity is largely hidden. (A good thing for the most part.)

I am sure I must be missing something. So let me pose the question: What, if anything, might ACM do more than it is already doing, to grow interest in and familiarity with computing and the computing profession?

Vinton G. Cerf, ACM PRESIDENT

a <http://www.acm.org/publications>
 b <https://myacm.acm.org/dashboard.cfm?svc=acmblogs>
 c <http://www.acm.org/sigs>
 d <http://www.acm.org/education>
 e <http://women.acm.org>

f <http://www.usfirst.org/>
 g <http://www.campus-party.org/home-en.html>
 h <http://icpc.baylor.edu/>

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:10.1145/2408776.2408780

<http://cacm.acm.org/blogs/blog-cacm>

When Reviews Do More than Sting

Bertrand Meyer wonders why malicious reviews run rampant in computer science.



Bertrand Meyer
“The Nastiness Problem in Computer Science”

<http://cacm.acm.org/blogs/blog-cacm/123611-the-nastiness-problem-in-computer-science/fulltext>
August 22, 2011

Are we malevolent grumps? Nothing personal, but as a community, computer scientists sometimes seem to succumb to negativism. They admit it themselves. A common complaint in the profession is that instead of taking a cue from our colleagues in more cogently organized fields such as physics, who band together for funds, promotion, and recognition, we are incurably fractious. In committees, for example, we damage everyone's chances by badmouthing colleagues with approaches other than ours. At least this is a widely perceived view (“*Circling the wagons and shooting inward*,” as Greg Andrews put it in a recent discussion). Is it accurate?

One statistic that I have heard cited is that in 1-to-5 evaluations of projects

submitted to the U.S. National Science Foundation the average grade of computer science projects is one full point lower than the average for other disciplines. This is secondhand information, however, and I would be interested to know if readers with direct knowledge of the situation can confirm or disprove it.

More examples can be found in the material from a recent keynote by Jeffrey Naughton, full of fascinating insights (see <http://pages.cs.wisc.edu/~naughton/naughtonicde.pptx>). Naughton, a database expert, mentions that only one paper out of 350 submissions to SIGMOD 2010 received a unanimous “accept” from its referees, and only four had an *average* accept recommendation. As he writes, “*either we all suck or something is broken!*”

Much of the other evidence I have seen and heard is anecdotal, but persistent enough to make one wonder if there is something special with us. I am reminded of a committee for a generously funded CS award some time ago, where we came close to not giving the prize at all because we only had “good” proposals, and none that a committee member was willing to die for. The

committee did come to its senses, and afterward several members wondered aloud what was the reason for this perfectionism that almost made us waste a great opportunity to reward. We come across such cases so often—the research proposal evaluation that gratuitously but lethally states that you have “*less than a 10% chance*” of reaching your goals, the killer argument “*I didn't hear anything that surprised me*” after a candidate's talk—that we consider such nastiness normal without asking any more whether it is ethical or helpful. (The “surprise” comment is particularly vicious. Its real purpose is to make its author look smart and knowledgeable about the ways of the world, since he is so hard to surprise; and few people are ready to contradict it: Who wants to admit that *he* is naïve enough to have been surprised?)

A particular source of evidence is refereeing, as in the SIGMOD example. I keep wondering at the sheer nastiness of referees in CS venues.

We should note that the large number of rejected submissions is not by itself the problem. Naughton complains that researchers spend their entire careers being graded, as if passing exams again and again. Well, I too like acceptance better than rejection, but we have to consider the reality: with acceptance rates in the 8%–20% range at good conferences, much refereeing is bound to be negative. Nor can we angelically hope for higher acceptance rates overall; research is a competitive business, and we are evaluated at every step of our careers, whether

we like it or not. One could argue that most papers submitted to ICSE and ESEC are pretty reasonable contributions to software engineering, and hence these conferences should accept four out of five submissions; but the only practical consequence would be that some other venue would soon replace ICSE and ESEC as the publication place that matters in software engineering. In reality, rejection remains a frequent occurrence even for established authors.

Rejecting a paper, however, is not the same thing as insulting the author under the convenient cover of anonymity.

The particular combination of incompetence and arrogance that characterizes much of what Naughton calls “bad refereeing” always stings when you are on the receiving end, although after a while it can be retrospectively funny; one day I will publish some of my own inventory collected over the years. As a preview, here are two comments on the first paper I wrote on Eiffel, rejected in 1987 by the *IEEE Transactions on Software Engineering* (it was later published, thanks to a more enlightened editor, Robert Glass, in the *Journal of Systems and Software*). The IEEE rejection was on the basis of such review gems as:

► *I think time will show that inheritance (section 1.5.3) is a terrible idea.*

► *Systems that do automatic garbage collection and prevent the designer from doing his own memory management are not good systems for industrial-strength software engineering.*

One of the reviewers also wrote: “*But of course, the bulk of the paper is contained in Part 2, where we are given code fragments showing how well things can be done in Eiffel. I only read 2.1 arrays. After that I could not bring myself to waste the time to read the others.*”

This is sheer boorishness passing itself off as refereeing. I wonder if editors in other, more established disciplines tolerate such attitudes. I also have the impression that in non-CS journals the editor has more personal leverage. How can the editor of IEEE-TSE have based his decision on such a biased and unprofessional review? Quis custodiet ipsos custodes?

“More established disciplines.” Indeed, the usual excuse is that we are

Rejecting a paper is not the same thing as insulting the author under the convenient cover of anonymity.

still a young field, suffering from adolescent aggressiveness. If so, it may be, as Lance Fortnow has argued in a more general context, “*time for computer science to grow up.*” After some 60 or 70 years we are not so young any more.

What is your experience? Is the grass greener elsewhere? Are we just like everyone else, or do we truly have a nastiness problem in computer science?

Readers' Comments

This is only a problem for academics. In the real world (industry), the customers stand in judgment.

—Anonymous

I am a physicist but have entered CS and now publish in this field. I do notice the attitudes you describe and they scare me because I get the impression that every other computer scientist is very insecure. Rude comments from reviewers are common and editors seem not to care. But more so, it is common that reviewers are clueless and barely understand the paper they review. So, if one reviewer is rude and clueless, and two are knowledgeable and positive, then the editor still mainly listens to the clueless one, simply because a negative critique is more worthy than positive in this field...

—Anonymous

As a reviewer and as an author, I get the feeling (in some cases I actually know) that some of my (co)reviewers did one of two things: Had someone else less/ not qualified review the paper, without bothering to check the quality of the review; or reviewed the paper at the last possible minute, probably after several reminders from the program chair.

In either case, it is hard to get a fair review.

—Anonymous

I am from physics, it has its own share of nastiness, different from what you describe. Right now, I work in a research organization dominated by computer scientists and have written and reviewed some computer science papers. At the risk of sounding haughty (I do not mean to), I would say:

As you have mentioned, computer science is a relatively new field and physics far more mature. This not only means that computer science has more upstarts reviewing and writing papers, but that the quality of research varies from excellent to mediocre to rather poor. As opposed to physics or natural sciences, where almost all research in a field is of similar quality (with respect to maturity). Now you might say that is good or not good, I don't know.

Also, computer scientists have far more funds to publish and hold conferences (at exotic locations), leading in turn to lots more papers to write and review, and all the related rage. I wrote one paper in two years and reviewed maybe a couple every year while I was in physics. I do some reviewing/ writing activity every week in computer science research.

—Anonymous

One challenge this poses to program chairs is that we can be misled by nasty reviews versus genuine rejections, especially when the nasty guy works hard. A solution might be to publish reviewer stats, including how often a reviewer is the minority, average length of review, and so on. It might improve behavior if we know that poor numbers could brand us out of prestigious committees. We all want to be on program committees, and then act as if we can't be bothered and are too busy! This needs to change. Coming from industry, I know that sticks work better than carrots!

—Anonymous

Bertrand Meyer is a professor at ETH Zurich and ITMO (St. Petersburg) and chief architect of Eiffel Software.

interactions

EXPERIENCES | PEOPLE | TECHNOLOGY



interactions magazine's website interactions.acm.org, is designed to capture the influential voice of its print component in covering the fields that envelop the study of people and computers.

The site offers a rich history of the conversations, collaborations, and discoveries from issues past, present, and future.

Check out the current issue, look up a past prototype, or discuss an upcoming trend in the communities of design and human-computer interaction.

FEATURES

BLOGS

FORUMS

DOWNLOADS

interactions.acm.org

Association for
Computing Machinery



Life in Simulation

Computational models are tackling the complexity of biology, from single-celled microbes to human organs.

BIOLOGISTS ARE AWASH in data; the genomes of 18,840 organisms had been sequenced by mid-October 2012, according to the U.S. Department of Energy's Joint Genome Institute. And scientists around the world are trying to determine how to wring value from all that data, with projects studying how those genes interact with each other and the environment, how embryos develop, how toxins affect tissues, and why some cells become cancerous, among many other questions. With so much data, and with questions that hinge on understanding the complex interplay of multiple factors, computer scientists are working to develop software that can simulate the behavior of biological systems, from cells to organs to entire organisms.

"It's trivial to generate terabytes of data in a day or two," says Mark Isalan, a systems biologist at the Centre for Genomic Regulation in Barcelona, Spain. The bigger challenge is how to use all that data to address important questions.

Yet that volume of data also makes problems in biology more tractable for computer scientists seeking to model biological behavior. "Because we can measure the cells much more precisely, we actually have numbers we



The huge volume of data generated from genome sequencing technologies, like those used as part of the DOE's Joint Genome Institute, has inspired computer scientists worldwide to create software that can take that data and build computational models simulating the behavior of biological systems.

can give to models,” Isalan says. And once they have built accurate models, researchers can then change some of the model’s parameters and see what happens, which can help explain the mechanisms of disease, or suggest new targets for drugs to treat diseases.

Researchers from Stanford University and the Craig Venter Institute in Rockville, MD, have taken an important step toward getting value out of the huge datasets by creating what they say is the first comprehensive computational model of a living organism. They built a model of the bacterium *Mycoplasma genitalium*, a parasite that infects the human urethra. The scientists chose that microbe because it has only 525 genes, the fewest of any independently living organism; humans, by contrast, have around 25,000. “We really wanted to build something that was complete,” says Jonathan Karr, a doctoral student in biophysics at Stanford and lead author of a paper about the work. “Anything larger we felt would just be impossible at this point to build a complete model.” Even this model, the researchers say, is essentially a “first draft.”

To build their model, the team broke down the cell into various individual functions. They came up with

28 sub-models, each simulating a different process. One, for instance, modeled the replication of DNA, while another described the process of transcribing RNA, and another simulated how proteins produced by the cell folded into particular shapes. One advantage of this approach, says Karr, is that it makes sense from a software engineering perspective; “That’s the way programs are built,” he says. Another plus is that, instead of using one type of mathematical representation to describe the entire organism, it allowed the researchers to use the mathematical approach most appropriate to the activity of a particular module and to the amount of data available about that cellular function. One module might rely on ordinary differential equations, for instance, while another uses a Boolean model. “Different aspects of cell biology are not all characterized to the same level of detail,” Karr explains. “There are just certain aspects of cell life we know more about and others where we know less.” The team fed the model data gleaned from research literature about the bacteria and other similar organisms, and added additional information they generated in laboratory experiments.

To make all the sub-models work together, the researchers built a piece of software to plug them all into. The sub-models run independently for a short time, less than a second. But to emulate how biology relies on feedback loops, the sub-models are linked by 16 variable metabolic states that, taken together, represent everything going on in the whole cell. At each time step of one second, the sub-models take the states of those variables and use them to run their simulation, then make the revised values available to the other sub-models. Also at each time step, the computer estimates the amount of metabolic resources a given biological process would require, then allocates the cell’s total resources proportionally among the different processes. These steps—measure, calculate, share, repeat—run thousands of times until the simulated cell reaches the point where a real-life cell would divide into two, at which point the simulation is done. When the team ran their simulation, the computer produced results that matched those that had already been determined in lab experiments. But beyond that, says Karr, the simulation can also highlight inconsistencies in the data and suggest the existence of cellular functions

Milestones

Computer Science Awards, Honors

ACM ANNOUNCES DISTINGUISHED MEMBERS

ACM has named 41 of its members as Distinguished Members for their individual contributions and their singular impacts on the dynamic computing field. Their achievements have advanced the science, engineering, and education of computing, enabling a range of technologies that drive innovation in the digital age. The 2012 Distinguished Members hail from universities in Australia, Denmark, Italy, Korea, China, and the U.K. in addition to North America, and from leading corporations and research institutions around the world.

To view the full list of 2012 Distinguished Members, visit <http://distinguished.acm.org>.

SMITHSONIAN RECOGNIZES THRUN’S UDACITY WORK

Sebastian Thrun won the inaugural Smithsonian American Ingenuity in Education Award for his work with Udacity, an independent online education company he co-founded that provides high-quality education for free. Thrun, a roboticist, AI expert, Stanford research professor, and leading light at Google X, describes Udacity as an effort to democratize higher education. In accepting the award, Thrun said there is so much potential to do better than the classroom. “Higher education should be a basic human right; we should amend the Constitution,” he said. “You can give a man a fish and he has dinner for the night,

or you can *teach* a man to fish and he has dinner for the rest of his life. That’s what education is all about.”

ABET HONORS ZWEBEN

Stuart H. Zweben, an ACM Fellow and former ACM president (1994–1996), has received the 2012 Linton E. Grinter Distinguished Service Award from the Accreditation Board for Engineering and Technology (ABET). The award cites Zweben, a Professor Emeritus at Ohio State University’s Computer Science and Engineering Department, for “outstanding leadership in computing accreditation worldwide.” In the early 1980s, Zweben played a key role in ACM’s decision to join with the IEEE-CS to form the Computing

Sciences Accreditation Board, now known as CSAB.

LEROY RECEIVES MICROSOFT AWARD

Xavier Leroy, senior computer scientist at INRIA’s Paris-Rocquencourt research center, received the 2012 Microsoft Research Verified Software Milestone Award. Leroy was recognized for his role as architect of the CompCert C Verified Compiler. Said Judith Bishop, principal research director, computer science, at Microsoft Research, “Compilers are the basis for all the software we generate, and by ruling out compiler-introduced bugs, the CompCert project has taken a huge leap in producing strengthening guarantees for reliable critical embedded software across platforms.”

that are not yet recognized, pointing the way to new lines of research. “The model helps us reason about the things we as a field collectively don’t know about cell biology,” he says.

From Bugs to Brains

While the Stanford group focuses on simulating a whole organism, other researchers are concentrating on simulating an organ. Since 2005, the Blue Brain Project at École Polytechnique Fédérale de Lausanne, in Switzerland, has been developing a computer model of a brain. So far, they have built and run a representation of part of a rat’s cortex, consisting of 10,000 neurons. The researchers have asked the European Union to fund a 10-year, one billion euro project to create a functioning model of an entire human brain, with hundreds of millions of neurons that could be used to simulate neurological diseases or the effects of various drugs on the brain. At press time, an answer was imminent.

In Germany, the Virtual Liver Network consists of 70 research groups working to build a model that, while not fully duplicating the liver, represents the physiology of the organ, simulating biological functions at different levels, from activity within individual cells to the liver as a whole. Meanwhile, the U.S. Environmental Protection Agency (EPA) is working on a similar project, with the aim of being able to simulate the effects of drugs and environmental toxins on the liver. The agency also has a virtual embryo project to study how certain chemicals might cause birth defects.

The EPA needs such data to set regulations about what levels of exposure to chemicals should be considered safe for humans. To date, such levels are set based on data from animal models, but an animal study can take up to two years and cost millions of dollars, says Imran Shah, a computational systems biologist at the EPA’s National Center for Computational Toxicology in Research Triangle Park, NC, who works on the virtual liver. Further, he says, there is some question as to how closely the effects of chemicals in animals match what happens in humans; there are proteins whose increased production causes liver cancer in rats but not in people, for in-

“The model helps us reason about the things we as a field collectively don’t know about cell biology.”

stance. And results in animal studies may come from the high concentration of toxins used in tests, which may not replicate a real-life situation. “In most cases what the EPA cares about is long-term and very low-level exposure,” Shah says.

The approach Shah’s team takes is agent-based multi-scale modeling. They make models at various levels of organization—the molecular pathways within a cell, the cell as a whole, groups of cells, sections of liver. Like the Stanford work, the whole model is built as a series of modules, with each module acting as an autonomous agent. One module might be responsible for metabolizing a substance, another might affect blood flow through capillaries. The simulation focuses on a lobule, a functional unit of the liver containing roughly one million cells of various types, with a defined three-dimensional structure. Blood flows through the lobule, nutrients are exchanged, bile is excreted. The team simulates the activity of a single lobule in detail, then groups 20 or 30 of them together to build a larger model of liver function.

This kind of modeling is a different approach to toxicology than statistical modeling, which looks for associations between, say, a potential toxin and a negative result. “We try to think about it more in terms of a mechanistic level,” Shah says. Mechanistic modeling may not just reveal that a chemical has an ill effect, but lead to a greater understanding of why.

The project is far from complete. The EPA team presented its first proof of concept this year, running simula-

tions on 10 virtual individuals with 10 virtual livers, but Shah said there needs to be much more refining of the biological information that goes into the models. One challenge that remains is verifying that what the model shows is a valid representation of the real world; how do you test the computer’s prediction against actual lab results when you cannot do these experiments on humans?

Karr would like to move from his bacterium model to more complex organisms, starting perhaps with yeast and then moving to simple multicelled organisms like worms, and on up the scale of complexity from there. Such models could allow synthetic biologists to design and engineer organisms, such as microbes that efficiently convert biomass into fuel or pharmaceuticals. And they could play a role in personalized medicine, allowing doctors to prescribe the best treatment based on an individual’s own genome and history. That will demand a lot of work, both gathering the biological information and figuring out the best computational approaches. “We eventually need to be able to understand how you get from a person’s DNA to the behavior of a human being,” Karr says. “And we’re going to need very detailed models to be able to do that.”

Further Reading

Karr, J.R. et al.

A whole-cell computational model predicts phenotype from genotype, *Cell* 150, July 20, 2012.

Wambaugh, J. and Shah, I.

Simulating microdosimetry in a virtual hepatic lobule, *PLoS Comput Biol* 6, 4, Apr. 22, 2010.

Isalan, M.

A cell in a computer, *Nature* 498, Aug. 2, 2012.

Wambaugh, J. and Shah, I.

Virtual tissues in toxicology, *J Toxicol. and Environmental Health, Part B*, 13, 2010.

Markum, H.

How do neurons connect to each other? Blue Brain Project opens new insights, Sept. 17, 2012; <http://www.youtube.com/watch?v=ySgmZOTkQA8/>

Neil Savage is a science and technology writer based in Lowell, MA.

Revving the Rover

The new Mars rover has attracted plenty of attention for its planetary gymnastics, but the big breakthroughs are under the hood.

WHEN NASA'S ROVER Curiosity touched down on the surface of the red planet on August 6, 2012, the cheering in Mission Control was soon echoed by a prolonged burst of public euphoria. Crowds gathered in Times Square at one in the morning to watch the landing on a giant TV screen, while millions of Web users blogged, tweeted, and otherwise applauded the embattled space agency's continuing ability to pull off Big Things.

Much of the Internet chatter centered on the acrobatic Sky Crane maneuver, in which the landing capsule morphed like a Transformer into a rocket-powered hovercraft to ease its precious cargo onto the planet surface. While the landing made for great online theater, those arresting images may also have diverted attention from several other important, though admittedly less telegenic innovations.

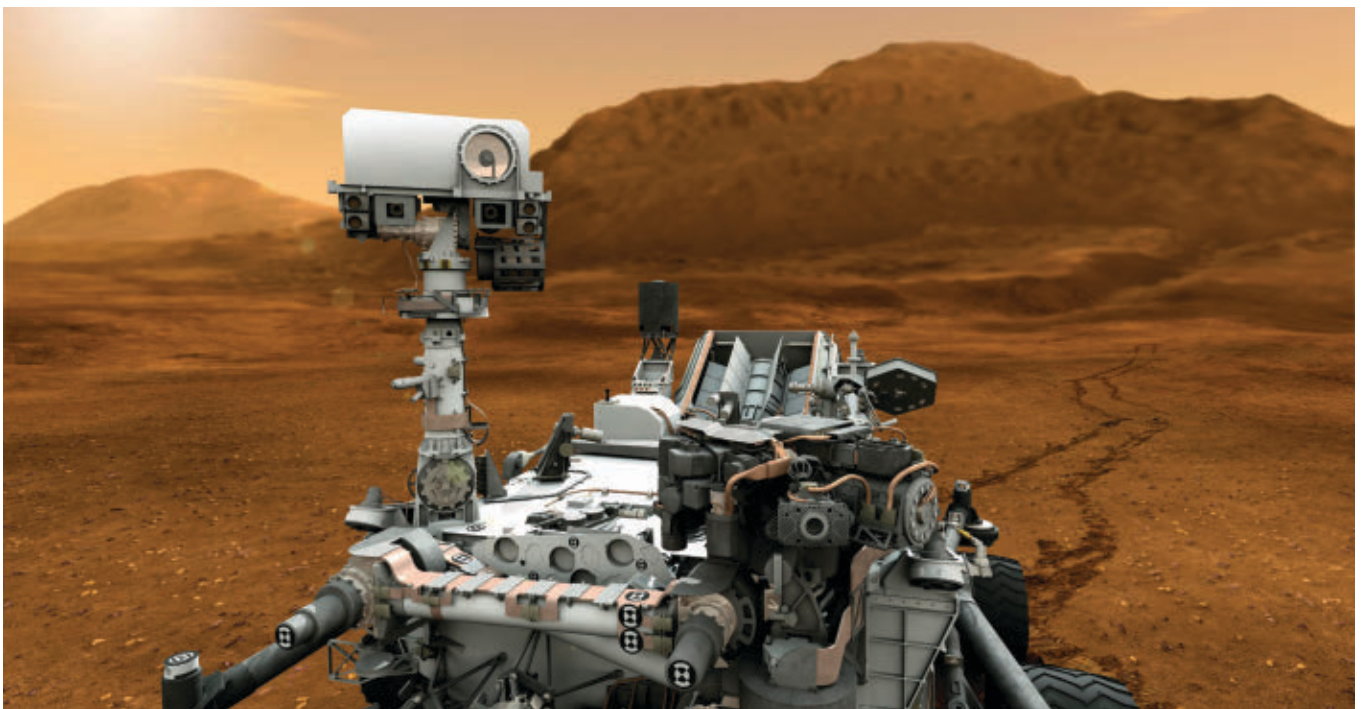
The Curiosity mission (formally known as the Mars Science Laboratory, or MSL) may also mark the end of an era for NASA, as planetary exploration approaches a level of engineering complexity that may call for fundamentally rethinking the design and architecture of future robotic missions.

"This represents the arc of an engineering process that really started in the 1960s," says Rob Manning, chief engineer of the Mars Exploration Program at NASA's Jet Propulsion Laboratory in Pasadena, California.

The earliest NASA missions of the 1960s and 1970s relied on highly distributed systems, with computing power resident on multiple devices, largely due to limitations in processing power. Starting with the Mars Pathfinder mission in 1996, however, the agency started to embrace a more centralized model, concentrating most computing tasks onto a single onboard computer.

While the basic contours of each rover mission have stayed roughly the same since then—namely, fly a spaceship to Mars, land a wheeled vehicle, then collect data while driving around the planet surface—the data gathering requirements have grown progressively more sophisticated with each mission.

The 1996 Sojourner rover was content to snap photos and perform x-ray spectrometry on a few rock samples within about 40 feet of the landing site. For the 2004 mission, the team gave the Spirit and Opportunity rovers considerably more autonomy, equipping them with a new software system dubbed Autonomous Exploration for Gathering Increased Science (AEGIS) that allowed the rovers to select potentially interesting research targets without requiring direction from Earth-bound controllers. Curiosity takes that autonomy several steps further, moving far and wide—powered by a plutonium-fueled nuclear engine—



NASA's rover Curiosity landed on Mars last August carrying almost 2,000 pounds of state-of-the-art scientific instruments.

while carrying an arsenal of 10 different scientific instruments, including cameras and imaging equipment, environmental sensors, and sophisticated sampling tools. Curiosity weighs in at nearly 2,000 pounds (compared to Sojourner's lithe 23 pounds).

Managing so much onboard equipment constituted an enormously difficult hardware and software design challenge. As the scientific requirements have grown more elaborate, the team has discovered the downside of centralized computing.

"We have one set of requirements for cruise, one for landing, one for on the surface," explains Manning. "So we have all this extra hardware and interfaces—and now we have to lug it all around."

While the all-in-one approach makes for a much bulkier machine than previous rovers, the complexity of the software stems primarily from the rover's high degree of autonomy, demanding millions of lines of code that would allow the rover to navigate the planet surface, identify and react to potential hazards while collecting samples, aim precision-targeted laser beams onto rocks several meters away, and communicate with Earth via its interplanetary ISP, NASA's prodigious Deep Space Network.

Managing so many discrete functions on the same machine demands a high level of functional decomposition, so that different routines can take over the system at appropriate times without compromising other essential features. As a result, the engineering team had to think carefully about issues of memory allocation and fault tolerance, as well as managing a bewildering array of input and output devices.

"Software grows exponentially fast in this domain," says Gerard Holzmann, head of the Laboratory for Reliable Software at NASA's Jet Propulsion Laboratory, an organization formed in 2003 to improve the reliability of mission-critical software. Indeed, with each successive mission to Mars, the size of the onboard flight code has more than doubled. While software engineers have long understood that software packages often grow over time—expanding to take advantage of faster processors and additional storage space, for example—that problem

The Curiosity mission may mark the end of an era for NASA.

gets further exacerbated by the complexities of safety-critical systems like space exploration.

"Managing the development of a few million lines of critical code carries very different challenges from the development of a few thousand or even a few hundred thousand lines," says Holzmann.

To cope with the scale of the MSL challenge, the team introduced several important new software reliability initiatives, including the design of a new Institutional Coding Standard that, while requiring relatively few strict rules, was designed to support automated compliance verification, allowing the team to run a nightly check on every new build. The team also introduced a new peer code review process and "scrub" tool integrated with a suite of static source code analysis tools including well-known commercial testing and analysis tools like Coverity, Codesonar, and Semmlle, as well as Uno, a source code analysis tool that Holzmann had developed several years earlier while working at Bell Labs.

The static source code analyzers played a critical role in the software development process, allowing the team to ferret out risks of data corruption, race conditions, or deadlocks.

"A good static analyzer is very much like employing an additional, very conscientious and tireless developer on your team," says Holzmann. "It's an extra set of eyes that never tires of pointing out new subtle flaws."

The team also had a secret weapon on hand in the form of Holzmann's Spin logic model checker, which he developed over a period of several decades in his previous job at Bell Labs. The system targets the formal verification of multithreaded software written

Sensor networks

WSNs Head to Himalayas

At press time a team of scientists was heading to the Himalayas to deploy innovative wireless sensor networks (WSNs) in several landslide-prone regions of the world's tallest mountain ranges to provide real-time warnings for often deadly deluges.

The development of these WSNs began about five years ago when an interdisciplinary team of researchers from Amrita University, Kerala, India, combined efforts to design warning systems that would alert people living in landslide-prone areas. The team was made up of computer scientists, Earth scientists, and energy experts and was led by Maneesha Sudheer Ramesh, a founding member of ACM-W India.

The warning system uses WSN technology to issue real-time warnings up to 24 hours prior to an impending landslide, thus facilitating evacuation and disaster management. During the creation of the WSN, Ramesh and her team built a working lab capable of mimicking a landslide. Published papers about the work intrigued the scientific community as well as the university's chancellor, Sri Mata Amritanandamayi Devi, who was the first to proclaim the technology worthy of real-world applications. "Let us actually deploy the wireless sensor network in the field and enable it to save lives."

The first WSN system was deployed in Munnar, Kerala, in June 2009, and has successfully delivered a high level of safety to citizens living in at-risk areas. While the team faced several challenges, including making the system energy sustainable during severe monsoon rains, Ramesh said she was surprised at the "immense amount of psychological safety and security the system seems to have given to the local population, which we never expected since the local population is not that tech savvy."

Last December, the team was awarded the top prize for rural innovation by the Government of India.

—Diane Crawford

in C, the language used for about 96% of all spacecraft software.

“A mission failure often has multiple small triggers that combine in unsuspected ways,” he says. “By meticulously fixing the small, relatively benign issues with the same determination as the larger issues, we make sure that serious problems become much less likely.”

Those problems are further exacerbated by the organization’s now deeply rooted commitment to a centralized computing architecture. Looking ahead, Manning thinks the NASA team will need to rethink its architectural approach for the next generation of robotic flight missions. “Going forward, I would take a more distributed approach,” he says.

For now, the team will continue to fine-tune Curiosity’s work from a distance over the next several months. But no matter how well the software works, they know full well that space exploration is an inherently unpredictable business—especially on Mars, where wild temperature swings and changes in atmospheric pressure can ruin delicate scientific instruments. To control the onboard temperature, the engineering team developed thermal “catcher’s mitts” (as Manning describes them) on the back of the machine, consisting of liquid freon pumped through a closed loop and warmed from the hot plutonium rocks that power the rover.

In order to model as many different scenarios as possible, the team is constantly running so-called soft simulations with dedicated machines analogous to the onboard RAD750 machine. The team also maintains a full replica of the rover on Earth in a testbed environment to troubleshoot problems and rehearse potential future maneuvers. If all else fails, the rover also carries a fully redundant version of its onboard computer, ready to swap in at a moment’s notice in case of system failure. “We work hard to make sure the vehicle doesn’t come to its knees if it has a small complaint,” says Manning.

For all its autonomy, Curiosity still depends heavily on regular communication with its handlers back on Earth. To stay connected with Mission Control—up to 240 million miles, or 21 light minutes away—it relies on a sophisticated interplanetary infrastruc-

For all its autonomy, Curiosity still depends heavily on regular communication with its handlers back on Earth.

ture consisting of Earthbound radio transmitters and receivers, and a constellation of satellites orbiting Mars.

NASA put the initial Mars communications system in place with the Mars Global Surveyor mission communicating to the Spirit rover in 2004. Since then, it has made steady improvements to mitigate the ongoing problems of data loss, such as space-link noise, interfering spacecraft, and unpredictable technical problems at the relay spacecraft and ground stations.

Complicating matters further are the limited transmission windows between Mars and Earth due to orbital constraints; usually Curiosity can connect to the network only 2–4 times per sol (astronomer-speak for a Martian day), transmitting an average of 64 megabytes/sol broken into packets, Internet-style.

“We are bandwidth limited,” says Sandy Krasner, a NASA software systems engineer who has been working on the Mars project for the past 10 years, “so we have to optimize the use of our downlink as much as possible.”

Given the high cost of retransmitting data, the network is designed to focus on error detection and correction, and maximizing loss tolerance. The system sets a maximum file size of one quarter of a megabyte on command files sent to the spacecraft; larger files are broken into smaller datasets and concatenated onboard. To ensure the integrity of data received by the rover, the system also detects and corrects for errors at multiple levels. Data is transmitted in 56-bit blocks assembled into variable-length frames up to 1 kilobyte.

The team also tries to tolerate faults in data received from the spacecraft, accepting partial transmissions of image data, for example, where an occasional pixel may get lost in space.

NASA is now working on a more distributed network protocol known as Disruption-Tolerant Networking (DTN) that distributes data across a network of nodes so that any delays or transmission failures can be corrected quickly by retransmitting the data. NASA hopes this architecture will make future interplanetary communication more efficient.

This ongoing network connectivity enabled the programming team to keep tweaking the rover’s software well after the mission’s launch date on November 26, 2011, sending updates to the onboard computer using a relatively low-tech solution: compressed binary files.

Last June, two months before landing, the team sent up its final in-flight software update while the capsule was hurtling through space at 13,000 miles per hour.

Manning remembers the satisfaction of looking on from a distance of 20-odd light years as rover installed the software and restarted, ready to strike out for parts unknown. “Boot it up and away we go.” □

Further Reading

Sky Crane Video;
<http://www.youtube.com/watch?v=N9hXqzkH7YA>

G.J. Holzmann
The Spin Model Checker—Primer and Reference Manual, Addison-Wesley, Reading MA, 2004, ISBN 0-321-22862-6.

Mars Science Laboratory (NASA site)
http://www.nasa.gov/mission_pages/msl/index.html

G. Groth
Software on Mars, Communications of the ACM 55, 11 (Nov. 2012), 13–15; <http://cacm.acm.org/news/156591-software-on-mars/fulltext>

B. Cichy
2010 Workshop on Spacecraft Flight Software, California Institute of Technology; <http://win-dms-ms1.caltech.edu/five/Viewer/?peid=476727664f1b4d8390d3ab37670ababd>

Alex Wright is a writer and information architect based in Brooklyn, NY.

© 2013 ACM 0001-0782/13/02

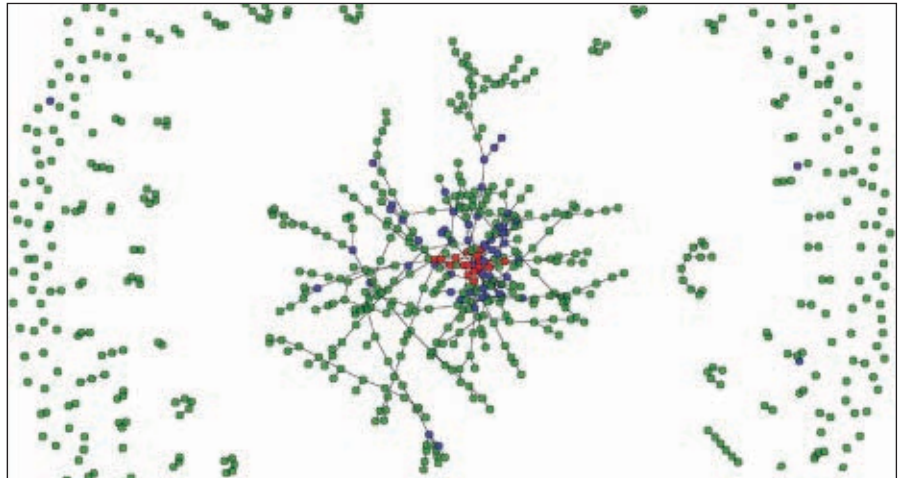
A New Model for Healthcare

Computer modeling is radically redefining healthcare and epidemiology by providing new tools for understanding the impact of different intervention strategies.

STEP INSIDE BRANDON Marshall's lab at Brown University and you get a glimpse into the unfolding future of epidemiology. Marshall, along with a multidisciplinary team comprised of IT specialists, software developers, mathematicians, healthcare analysts and others, are melding their expertise to take HIV policymaking to a new level. In a world filled with what-if questions, the assistant professor of epidemiology is supplying answers by building a model that in many respects resembles a consumer-oriented simulation game such as *The Sims*.

For example, Marshall can choose to view a fictional person based on particular characteristics. Overall, approximately 150,000 life courses reside within the model, which is focused on New York City. By selecting one agent and observing the interactions with other agents—including how they approach sex and drugs and how officials and policymakers approach issues such as the distribution of condoms and sterile needles, it is possible to understand behavior and choices in a far more complete and realistic way. “We can track individuals over time, which is really novel and exciting,” he says.

Marshall is one of a growing number of researchers turning to computer modeling to go where researchers have not gone before. For most of human history, understanding the impact of a disease or health problems was nothing less than guesswork. When a new virus or bacterial infection popped up, public officials did their best to assemble data, extrapolate on it and establish a course of action. “The problem,” observes Marshall, “is that healthcare and public policy experts have had limited knowledge and visibility into all the factors and variables.”



Last summer, Brown University's Brandon Marshall unveiled a computer program calibrated to model accurately the spread of HIV in New York City over a decade and to make specific predictions about the future of the epidemic under various intervention scenarios.

However, by plugging into massive datasets and tapping into today's computing power—the list includes mobile tools, social media, and crowdsourcing to track movements and behavior in real time—it is possible to create far more realistic forecasts and what-if scenarios. The impact of computer modeling on decision-making and public policy is nothing less than revolutionary. Ruben Juanes, an associate professor at MIT, also turned to algorithms and advance models “to understand extremely complex issues in ways that weren't possible in the past.”

By the Numbers

The idea of using computers to analyze health data is nothing new, of course. In the 1950s, researchers—including epidemiological and public health officials—began exploring ways to transform data into actionable models. Unfortunately, a general lack of computing power and far less sophisticated software imposed severe limits on the breadth and depth of analysis that could take place.

“Traditionally, researchers have relied on a reductionist approach to problem solving,” points out Patricia Mabry, senior advisor in the Office of Behavioral and Social Sciences Research at the National Institutes of Health (NIH). However, this approach—known as a randomized controlled trial—typically addresses only one small aspect of the overall problem. Reductionism essentially attempts to condense and simplify issues in order to make the process more manageable. Controlled studies focus on different groups and different assigned or measured variables. Both of these techniques rely on a relatively narrow focus.

It was not until the 1990s that the first robust computer models began to appear. The addition of the Internet and big data analytics has revolutionized computer modeling over the last decade. Today, many new data points exist, including databases and metrics gleaned from mobile geolocation data and social media. In the healthcare arena, researchers now use computer model-

ing to examine the effect of different HIV policies, emergency response scenarios surrounding a poison gas attack, how eating and exercise impact obesity and healthcare costs, and more.

Comprehensive simulation modeling is a key to effective decision-making, particularly in today's cost-conscious environment, states Peter Aperin, M.D., vice president of medicine for Archimedes Inc., a San Francisco, CA, company that develops full-scale simulation models of human physiology, diseases, behaviors, interventions, and healthcare practices for the likes of Kaiser Permanente and the Robert Wood Foundation.

Aperin says these techniques are useful for examining a wide range of issues: everything from obesity and diabetes to heart disease treatment options. "What if we change treatments? What if we use different interventions? What if different interventions or treatments have different side effects or cause different behavioral changes? We are able to view how an almost infinite number of decisions affect downstream health outcomes and costs." In fact, it is possible to visualize these complex datasets rather than poring over an endless stream of statistics and numbers.

Moreover, researchers can select specific groups—men, women, gays, those who take certain medications, or display high or low compliance rates and much more—and watch these simulations play out as agents interact with others and change their behavior over time. These virtual people influence each other, just as they do in the real world. It is also possible to toss new factors or variables into the equation, such as a virus that mutates or a new pharmaceutical drug, and watch events unwind in a completely different way. After researchers run a simulation a number of different ways they begin to view patterns and trends that provide valuable clues about public policy strategies and outcomes.

Marshall's HIV research is a perfect example. "The modeling allows us to combine data in very interesting ways that would be almost impossible empirically," Marshall explains. "We can combine datasets based on things like gender, sexual orientation, drug use, drug treatments and associated behavioral outcomes, access to drug abuse

Comprehensive simulation modeling is a key to effective decision-making in today's cost-conscious environment.

treatment programs and more. The model lets us see how decisions support one another or how an approach or program could prove detrimental." Marshall's model is designed to examine different approaches and policy decisions over a 20-year span.

Consider "Agent 89,425," who is male and has sex with men. He participates in needle exchanges, but according to the probabilities built into the model, in year three he begins to share needles with another drug user with whom he is also having unprotected sex. In the last of those encounters, Agent 89,425 becomes infected with HIV. In year four he begins participating in drug treatment and in year five he gets tested for HIV, starts antiretroviral treatment, and reduces the frequency with which he has unprotected sex. Because he takes his HIV medication without exception, he never transmits the virus further.

The research has already yielded some remarkable insights. For example, Marshall projects that with no change in New York City's current programs, the infection rate among injection drug users will be 2.1 per 1,000 in 2040. Expanding HIV testing by 50% would drop the rate only 12% to 1.9 per 1,000; increasing enrollment in drug treatment programs by 50% would reduce the rate 26% to 1.6 per 1,000; providing earlier delivery of antiretroviral therapy and better adherence would drop the rate 45% to 1.2 per 1,000; and expanding needle exchange programs by 50% would reduce the rate 34% to 1.4 per 1,000. However, adopting all four tactics would cut the rate by more than 60%, to 0.8 per 1,000.

These types of analyses/dependencies are routine in large data analysis in other fields. Is this true in healthcare or are the actors just learning this now? Or is Marshall's model something new and better than what already exists?

Putting Models to Work

The appeal of advanced computing modeling—referred to by Mabry and her NIH colleagues as *systems science methodologies*—is that it allows researchers far greater latitude to address the complexity of real-world phenomenon and "and to investigate what-if scenarios that cannot be studied in the real world due to time, money, ethical, or other constraints," Mabry says.

This is no small matter. Complexities of the real world include making sense out of bidirectional relationships where one variable affects another and vice versa. Because changes in the variables feed off one another, such relationships have the potential to, as Mabry puts it, "generate *vicious cycles* in which we observe things deteriorating rapidly or *virtuous cycles* in which we observe a situation improving rapidly."

Because the underlying cause of such situations is not always apparent to the naked eye, policymakers may actually make bad situations worse by applying a remedy to the wrong place in the system. "The goal of using systems science methods is to understand how the various components that make up a system interact and affect each other to produce an outcome. These methods excel at identifying nonlinear relationships, and time-delayed effects as well as interdependencies," Mabry explains.

At MIT, researchers in the Department of Civil and Environmental Engineering, led by Ruben Juanes, are applying seemingly incongruous methods to understand contagion dynamics through the air transportation network. Presently, the team is attempting to understand how likely the 40 largest U.S. airports are to influence the spread of a contagious disease originating in their home cities. The project could help determine how fast a virus might spread and appropriate measures for containing the infection—from quarantining individuals to closing airports—in specific geographic areas. This information could

also aid public health officials in making decisions about the distribution of vaccinations or treatments in the earliest days of contagion.

In order to predict how fast a contagion might spread, researchers are examining variations in travel patterns among individuals, the geographic locations of airports, the disparity in interactions among airports, and waiting times at individual airports. Juanes, a geoscientist, has tapped past research on the flow of fluids through fracture networks in subsurface rock to build an algorithm for the current task. Moreover, the team plugs in cellphone usage data to understand real-world human mobility patterns. The end result is “a model that’s very different from a typical diffusion model,” he says. It is plugging in more data—and better data—to create a more robust model than has ever before existed.

Archimedes’ Peter Alperin says that today’s models are aiding and speeding policy decisions in ways that were unimaginable only a few years ago. The NIH, government agencies, pharmaceutical firms, and healthcare organizations use these models to help build more effective policies or develop treatment strategies or new medicines. Last year, the firm began building models for the U.S. Food and Drug Administration (FDA) to better understand clinical trials evaluating weight loss medications. The data is being used to better understand the benefits of weight loss against the long-term risks of cardiovascular outcomes in patients treated with weight loss drugs.

Nevertheless, computer modeling is not a fix-all, says Sandro Galea, chair of the Department of Epidemiology at Columbia University. Among other things, he has examined how policy decisions affect social problems ranging from obesity to how large-scale disasters and trauma affect mental health among various demographic groups. In the latter scenario, for example, modeling helps identify who is at greater risk and what types of treatment and services can help reduce mental illness.

However, all models are built on assumptions and have some flaws and errors. Indeed, there is no standard for how to build an effective computer model or to establish confidence in what a model produces. Mabry and her

partners at NIH—a major provider of research grants—are now focusing on model “verification, validation, and uncertainty quantification.” They hope to collectively produce some guidance for model builders, as well as those reviewing journal manuscripts and grant applications. Without this, the fledgling field will continue to produce models that range widely in quality, she says.

Yet, computer modeling continues to evolve and gain acceptance. “Computer modeling isn’t a crystal ball,” Mabry concludes. “But it is helping to illuminate the complexity of health and social problems—along with potential remedies. Success is ultimately dependent on culling huge amounts of data about the population, developing good algorithms, and harnessing the success of supercomputers to make sense of complex relationships. This information can then be used by public policymakers to do their job.”

Further Reading

Auchincloss, A.H., Gebreab, S.Y., Mair, C. and Diez Roux, A.V.

A review of spatial methods in epidemiology, 2000–2010, *Annual Review of Public Health* 33, 107-122; <http://www.annualreviews.org/doi/abs/10.1146/annurev-pubhealth-031811-124655>

Marshall, B.D.L., Paczkowski, M.M., Seemann, L., Temalski, B., Pouget, E.R., Galea, S. and Friedman, S.R.

A complex systems approach to evaluate HIV prevention in metropolitan areas: Preliminary implications for combination intervention strategies, *PLoS*, 2012; <http://www.plosone.org/article/info%3Adoi%2F10.1371%2Fjournal.pone.0044833>

Meyer, J., Ostrzinski, S., Fredrich, D., Havemann, C., Krafcyk, J. and Hoffmann, W. Efficient data management in a large-scale epidemiology research project, *Computer Methods and Programs in Biomedicine* 107, 3, Elsevier North-Holland, NY, 2012.

Mysore, V., Narzisi, G. and Mishra, B. Emergency response planning for a potential Sarin Gas Attack in Manhattan using agent-based models, agent technology for disaster management, Hakodate, Japan, 2006; <http://www.cs.nyu.edu/mishra/PUBLICATIONS/06.sarin.pdf>

Sobolev, B., Sanchez, V. and Kuramoto, L. *Health Care Evaluation Using Computer Simulation*, Springer, 2012.

Samuel Greengard is an author and journalist based in West Linn, OR.

© 2013 ACM 0001-0782/13/02

ACM Member News

DAVID PATTERSON'S 'BIG DATA' PROJECT TAKES AIM AT A CANCER CURE



David Patterson and his team have been working for over a year on what he describes as an odd sort of

project for a computer scientist—building a software pipeline for cancer genomics that is faster, cheaper, and more accurate than ones that already exist.

Patterson, a former ACM president who has been a computer science professor at the University of California Berkeley since 1977, recalls an application was needed for the university’s new AMPLab, which integrates Algorithms, Machines, and People to make sense of “big data.”

“A problem in academics is that data is either small and interesting or big and dull,” he says. “Interesting big data is usually proprietary. But, in the case of cancer genetics, we knew there would be lots of data and a really important use for it—helping discover treatments that might put an end to what is become the second leading cause of death in the U.S.”

The pipeline uses The Cancer Genome Atlas, a repository of five petabytes of data containing the genetic sequencing of thousands of cancer tumors. It is expected to grow to millions—along with what treatments were given to patients to cure those tumors, and the outcomes. “The ultimate goal is that, by sequencing the genome of a cancer tumor, doctors will be able to prescribe a personalized, targeted therapy to stop a cancer’s growth—or cure it,” he says (see <http://nyti.ms/rJOjeS>).

Patterson, no stranger to fighting diseases, has raised over \$200,000 to fight multiple sclerosis after his wife was, fortunately, misdiagnosed with the disease. “Helping people fight both cancer and MS are worthwhile causes that work against the unfortunate stereotype of the uncaring computer scientist,” he says.

—Paul Hyman

Privacy and Security

The Tangled Web We Have Woven

Seeking to protect the fundamental privacy of network interactions.

THE LAST GENERATION is being born whose brains will develop independently of the Net. From now on, the way the Web works will play a dominant role in the socialization of the human race. But because we have built Web infrastructure without considering privacy, we are also endangering our basic freedoms. We are on the verge of eliminating forever the fundamental right to be alone in our thoughts.

At the beginning of the sixteenth century, moveable-type printing created the experience of private reading, and with it the Western idea of the individual self freely developed, self-made through a private process of reading and thinking. In religion, this led to the revolutionary adoption of individualist forms of Protestant Christianity. Secular society adopted the scientific method, and with it began radically improving the human social condition. The opening of learning also enabled the gradual transformation of the Western political landscape toward democratic self-government and the

constitutional protection of freedom of thought.

The Net should now universalize that process throughout the human race, should make it possible for every person on Earth to read, watch, listen, and participate in every form of learning and culture, everywhere, without discrimination between rich and poor, old and young, male and female. This truly universal learning system would immeasurably improve the welfare of humankind. But if we do not protect the fundamental privacy of network interactions,

We are on the verge of eliminating forever the fundamental right to be alone in our thoughts.

if we permit not only active surveillance but also extensive data mining of personal information in the Net, we will not achieve that promise. Indeed, if the Net is not engineered to protect privacy, it will instead become a jail for the human body and the human soul.

We are failing at present because our Net is being used to spy on us, constantly, as we use it to enrich our lives. The innovations in surveillance have come from industry. Record-keeping about how we use the Net—what we search for, what we read, who we contact—is intensively and instantaneously “mined” for its value to those who want to sell us something. What we share with our friends and family, even the content of our email and other private communications, is scrutinized to the same end by companies that offer us “services” in return for access to our private data. All this data, assiduously gathered by businesses seeking profit, no matter how responsibly they manage it, is also at the disposal of any government capable—by law, force, or fraud—of gaining their cooperation.



Beyond the data itself lies the new mathematics of inferring from it. “Data mining,” which now politely refers to itself as “data science,” is a new subdiscipline of statistics, directed at using all this individually identifiable and aggregated behavioral data to predict human social action. Whether one is selling pharmaceuticals, toys, advertising placement, or a political candidate, data science is now using our personal data to help the seller identify, pursue, and persuade us. Our consumption supplies information that can be used to read our minds.

The situation is made still worse because we are rapidly adopting personal service robots that are not working exclusively in our interests. Unlike the robots living intermixed with humans in the science fiction of our childhoods, these robots have no hands and feet—we are their hands and feet. They see what we point them at; they have ears to hear everything going on around us; they know our location all

the time. These robots we call smartphones and tablets often contain software we cannot read or understand, much less change. We do not control them; rather, they offer others the opportunity to control us.

Development in the private market of technologies to surveil, predict, and influence individuals through the Net has of course drawn the attention of states. Governments are rapidly moving, to the fullest extent of their differing means, to harness the power of big personal data to improve their social control. No matter what your politics, somewhere in the world, right now, a government of whose principles you completely disapprove is beginning to use the Net to locate support, influence the population, and find its enemies. Everywhere in the world, from now on, governments that become tyrannical will have immensely powerful new tools for remaining permanently in power.

This privacy crisis is ecological. The unintended consequences of tiny in-

dividual activities, aggregated over the vast scope of the Net, are producing a threat to our common human interests on a global scale.

Fortunately, because the parts of this crisis are all our creation, we can remedy the problem. We need to rebuild the operating software of the Net in keeping with certain ethical principles. This does not mean forcing people or businesses to change what they are presently doing. It means providing the equivalent of green technologies, and helping people shift to them.

First, then, we need to build responsible replacement software, providing existing functions in ways that respect users’ privacy, to replace systems that are hazardous to privacy. Current webmail and social networking services, for example, put all their users’ communications with their respective social circles inside huge centralized databases maintained by the service operator, who in return for doing the storing and providing sophisticated access services to users,

gets the right to mine the data, which is now centralized and vulnerable to government acquisition.

But email and the Web are by design *federated* services, in which individual servers can provide storage and access services cheaply, securely, and with near-perfect reliability for individual users. Users began using centralized services that hurt their privacy because they gained tangible convenience at no apparent cost. No one knew how to run her own mail server or Web server, and we did not make it easy to learn. But we can—and we should—help people to use free software and a coming flood of inexpensive “personal server” hardware to make personal privacy appliances.

The FreedomBox Foundation I am currently advising is an example of an attempt in this direction, making free personal privacy software for creating such appliances. Small, inexpensive, power-miserly devices you just plug in and forget, they keep your communications private, help you navigate the Web without being spied on, and let you share with the world, safely. Let me get technical for a few sentences to describe how.

Much of the implementation of such a software stack involves using existing free software tools. A privacy proxy located in the router between a user’s smartphone or PC browser and the public Net can remove advertising and Web bugs, manage cookie flow, and improve browsing privacy and security by providing “HTTPS everywhere.” Automating use of SSH proxies and personal VPNs can not only protect the privacy of Web access behind the FreedomBox used as a router, it can also provide secure communications and privacy-protected Web access from a mobile device used on untrusted networks away from home.

Some of the tools needed for personal privacy appliances are combinations of existing functionality. Combining a HTTPS Web server and a XMPP server with OpenPGP-based authentication, for example, along with a method for building the “web of trust” through exchange of public keys embodied in QR codes (the 2D barcodes that smartphones already scan) yields a method for secure text,

When we act to improve our own privacy we are also protecting the privacy of our children, our families, and our friends.


voice, and video chat that is easy for ordinary users to deploy. That in turn also easily extends to a method for secure communication with journalists and public media outlets for relaying video and audio recorded with mobile phones. Beyond our present stage of development lie the new tools we need to build, like federated social networking software that can smoothly and without disrupting the web of social sharing replace Facebook and similar “services,” that have imposed centralized storage, data mining, and control.

Soon, such privacy servers will be available to replace your home wireless router or other similar device at even lower cost, but with enormous overall social benefit. Think of them as personal coal-scrubbers that cost next to nothing and improve the atmosphere we all breathe.

But this is not all. We must also provide clear, factual, technical public education about privacy and “the cloud.” Currently basic technical information is either altogether missing from or else distorted in the public debate. We need to help people understand why they might be better off storing their personal data on physical objects in their possession rather than in other peoples’ data centers in “the cloud.” We should make the results of “data science” accessible to a public that will never interest itself in the mathematics.

We must help people think eco-

logically about privacy. Users do not recognize that their correspondents’ privacy is also reduced when they use a “free” email service that reads and data mines email sent and received. They do not realize that everyone in the photographs they post on centralized social networking services is being facially identified and tagged. That the social networking service’s operator has access to all those pictures and all the tags, and so does anyone with whom the operator “cooperates.” We need to explain that every little decision to give away one’s own information also gives away other peoples’. We can teach people that when we act to improve our own privacy we are also protecting the privacy of our children, our families, and our friends. If we help people around us to understand the effects their actions have on others, they will decide for themselves what changes they should make.

Untangling the Web, restoring privacy in what we do and anonymity in what we read, will not be easy. Many fine businesses will make a little less money if we do not offer all our personal data to be mined by intermediaries on their behalf. Governments—pretty much all governments of every stripe—are rapidly discovering how much real control they can get without showing their hands if they make use of the currently misconfigured, anti-privacy Net. A consensus of the great and the good against privacy is forming; the one against anonymity is already full-blown. Imagine how different our world would be if all the books in the West for the last half-millennium had reported their readers to headquarters, including informing the Prince or the Pope how many seconds each reader spent on each page. The book, which anyone could read to herself in the privacy of her mind, is being replaced by an appliance that tracks your reading for the bookseller, subject to the Prince’s subpoena. It will not be easy to save privacy. But if we believe in liberty, we have absolutely no choice. 

Eben Moglen (moglen@columbia.edu) is a law professor at Columbia Law School and the founding director of the Software Freedom Law Center in New York.

Copyright held by author.

Inside Risks More Sight on Foresight

Reflecting on elections, natural disasters, and the future.

MY PREVIOUS COLUMN, “The Foresight Saga, Redux” (*Communications*, October 2012), began a discussion that is continued here regarding some lessons learned from the 2012 U.S. November elections. I also pick up on where I left off four years ago in my column “U.S. Election After-Math” (*Communications*, February 2009). In addition, I reflect on the collateral effects of Hurricane Sandy, along with the needs to anticipate and minimize the potential effects of other natural disasters more generally.

Election Integrity, Oversight, Accountability, and Auditing

State and federal roles in elections thus far have been rather inadequate, failing to provide any meaningful assurances that elections can be conducted without serious problems, especially where these roles have often become strongly politicized. It is clear that some sort of impartial oversight is necessary to ensure integrity throughout the entire election process—from beginning to end. At present, every step along the way is a potential weak link, with respect to accidental and intentional misuse as well as deceptive or otherwise biased practices that create voter confusion and inconvenience. It is also clear that much greater accountability is necessary, particularly in cases where rectification of egregious problems is difficult, or in some cases rendered essentially impossible, as a



Voting machines located under a tent at a temporary voting site in the Rockaways, NY, following Hurricane Sandy.

result of shortsighted legislation and regulations, inadequacies of proprietary systems, and the lack of foresight and planning for exceptional conditions such as clearly evident election irregularities and process disruptions.

However, these considerations were exacerbated by what happened in the northeastern United States in the week before the 2012 U.S. general election: Hurricane Sandy resulted in ensuing losses of power and Internet access, shut-downs of public transit and businesses, and losses of life and property. The federal, state, and local government responses were generally exceptional, although Election Day on the East Coast was severely complicated in many places as a result.

Various attempts were made to reduce the hardships that voters had experienced—by allowing for more early voting, extending polling place hours, accommodating voters whose polling places were without power or otherwise inaccessible, and actually issuing and counting many more provisional ballots. However, inherent weaknesses in the election process made some of the would-be fixes even more vulnerable to unfortunate disruptions and even willful misuse—such as last-minute changes in software, procedures, and even voting places. When a voter has neither electricity nor the ability to travel (no gas, no subways, or other transportation options), and when polling places with no power have to be

relocated, voting in person can become exceedingly difficult and confused by misleading reports of voting site unavailability—sometimes intermixing both real and bogus location changes. Furthermore, proposed emergency alternatives of voting by Internet or email without adequate preparation and concern for the possible risks, or even trying to print a ballot from some last-minute supposedly correct location on the Web, are likely to be problematic in the absence of electrical power, supposedly trustworthy computers, the rush to provide those alternatives without any real assurances, and so on.

One of the main goals for the conduct of elections should be to provide sufficient assurance throughout the entire process such that every loser and every voter who voted for any of the losers can justifiably believe that the elections were fair and justly evaluated—that is, that there were no events, circumstances, or externalities, accountable or otherwise, that might have altered the results.

From the perspective of the past Inside Risks columns and our periodic discussions of factors relating to election integrity, one of the most interesting aspects in the 2012 November elections was that the results of the Presidential race were definitive enough that they did not depend on the outcomes in larger states such as Ohio and Florida. If those results had been very close, it is quite likely that we would have seen prolonged law suits from both parties leading to the Supreme Court—irrespective of the perceived initial outcome. In Florida, the outcome of the presidential election was apparently not known officially for a week. In Arizona, it took two weeks to resolve three Congressional races because of the huge number of provisional ballots, all of which were ultimately counted after challenges by the losing candidates. Almost three weeks after the election, votes in 37 states and the District of Columbia were still waiting to be counted.

Overall for the election for all offices and ballot measures, numerous issues arose during the campaigning and the voting process. Examples related to voter registration, voter disenfranchisement, voter authentication, restrictions on early voting, shortages

of voting machines and trained election officials that resulted in huge lines in certain precincts, unsanctioned and unsupervised last-minute changes to proprietary election software, reported cases of vote flipping on touch screens in both directions, inconsistent party affiliations with unclear implications for straight-party voting, irregularities in issuing, validating, and counting provisional ballots, cases in which more votes were reported counted than ballots issued, disappearing ballots, inconsistencies in announcements of policies, deceptive practices, poorly defined policies for reviewing and definitively recounting close races, potentially risky emergency attempts at alternatives (noted earlier), along with many other factors such as the perception of even less visibility, accountability, and oversight for other than top races. The Supreme Court ruling that corporations are people as well as relaxed procedures on contributions also skewed the election processes, and gives the appearance of elections being bought.

Most of these problems were predictable. For many years, Inside Risks columns have reported issues with voting machines (disabled, failing, or miscalibrated touch-screens, erratic and nonreproducible behaviors, serious shortages of alternatives in times of failures, lack of accountability and audit capabilities) and election processes (for example, inadequate allocation of operative voting machines and provisional ballots, lack of adequate procedures for election integrity, reports of insider misuse and in some cases de-

Some sort of impartial oversight is necessary to ensure integrity throughout the entire electoral process—from beginning to end.

monstrable fraud by election officials), to name just a few. Several specific anomalies deserve mention here.

► Andrew Appel noted some serious irregularities in New Jersey, where the Lieutenant Governor issued a well-publicized directive permitting storm-displaced voters to vote by email—despite the state’s declared illegality of the announced directive as stated. Matt Blaze further warned that New Jersey’s emergency email voting could be “an insecure, illegal nightmare” (see <https://freedom-to-tinker.com/blog/appel/nj-lt-governor-invites-voters-to-submit-invalid-ballots/>).

► Voting system software was upgraded with “experimental” patches just a few days before the election in 39 counties in Ohio, bypassing normal election night reporting, and purportedly “fixing” problems.

► Alex Halderman demonstrated how easily existing voter registration addresses and other voter personal information in Washington state and Maryland could be accessed and changed online, by anyone else—based only on the ability to provide some publicly available personal information on the would-be victim.

► In addition to reports of on-screen vote flipping, machines in Covington, VA, mistakenly listed the Obama-Biden ticket as Republican, leaving open the question of what would happen under straight-party voting.

► Reports by Thom Hartmann and Sam Sacks (Truthout, The Daily Take blog) discussed claims by the Anonymous group regarding attempts to rig the presidential election in three states. Irrespective of the validity of those claims, it is clear that such efforts could succeed with relatively little evidence based on the fragility and lack of accountability in the existing proprietary election systems.

► ORCA, the Republicans’ high-tech program to dynamically monitor voting trends and identify potentially sympathetic voters, failed during the election.

If Internet-based and other remote computer or mobile-device enhanced voting is ever to take place in any widespread use, it deserves much greater scrutiny, accountability, and oversight—considering the risks of tampering, coercion, vote selling, and vote

buying. For example, see Barbara Simons and Douglas W. Jones, "Internet Voting in the U.S.," *Communications*, October 2012; See also Mark Halvorson and Barbara Simons, "Recount Roulette," *Huffington Post* (http://www.huffingtonpost.com/barbara-simons/voting-ballots-recount_b_2069192.html?utm_hp_ref=politics).

Above all, elections represent a collection of holistic problems that encompass not just technology but also everything else that is largely nontechnological—governments, policies, lobbyists, corruption, and political biases. For example, the U.S. Election Assistance Commission currently has no commissioners, and has been reduced to the efforts of a few staffers. Concerted efforts to disenfranchise voters seem to have succeeded in making voting much more difficult than it should be, and yet evidently resulted in some major efforts to counter them. The nontechnological aspects of achieving equal opportunity for voters seem to dominate the technological issues, which are themselves considerable.

In retrospect, unauditible proprietary paperless direct-recording voting machines (for example, with touch-screens or other non-keyboard inputs, but typically with no real assurance for system integrity or meaningful trustworthy audit trails) seem to be generally discredited by the security community, but nevertheless still used—irrespective of the risks. Similarly, proposals for casting ballots over the Internet all seem to ignore the risks of integrity compromises, denials of service, loss of privacy, and vote selling/buying. However, consensus seems to be emerging that the most sensible approach at the moment utilizes computer-scanned hand-marked paper ballots (even if obtained via the Internet, perhaps in the case of overseas voters). Such systems can achieve a measure of verifiability that is unattainable by the unauditible direct recording systems and by Internet voting—in part because they provide something tangible against which discrepancies and other irregularities can be evaluated. However, significant further research and development are needed, plus enforceable operational procedures directed at the realization that many

More attention needs to be devoted in the future to proactive planning for adversities.

of the risks in elections also lie far beyond the technology. Once again, the efforts to obtain pervasively fair elections are decidedly holistic.

Emergency Preparedness and Oversight

The effects that Hurricane Sandy had on the election on the East Coast (and elsewhere because of airport closures) remind us of the importance of trying to expect the unexpected and acting according to standards of preventive care. For example, climate change is now scientifically a reality, and needs to be confronted realistically. In addition, past hurricanes, earthquakes, tornados, and so on always tend to remind us that we do not devote enough attention to emergency preparedness. On October 29, 2012, Hurricane Sandy devastated shore areas of New Jersey and New York, with ocean surges destroying houses, disrupting travel, causing long-lasting power outages affecting millions of people (in some cases without power for weeks). Some landline and mobile telecommunications were shut down—with reports of failures of undersea cables as well. Wired and wireless Internet infrastructures were also affected, including some entire data centers. Payphones were suddenly in great demand. Various deaths were reportedly caused by the hurricane. Many organizations without off-site backup systems or enough emergency generators and spare fuel were seriously hindered in their efforts to recover. Some parts of the New York subway system were completely shut down for many days by flooded tunnels and damaged wiring. The PATH Trans-Hudson line from Newark to the World Trade Center was inoperable

for almost a month, and the line from Hoboken, NJ, even longer. Enough of the New York University Bellevue Hospital backup system was situated in a basement that flooded, necessitating evacuation of the hospital. Seventeen million gallons of water had to be pumped out from the basement of the hospital, although the pumps in the basement shorted out and were unable to feed the backup generators on the 13th floor!

Of course, only some of these problems were suggested by past experiences going back to the November 1965 New England blackout, but the effects of Sandy were in many ways unprecedented. However, the scale of the disruption probably exceeded the overall disruption during the ice storm of 1998 in Quebec and Ontario, when the power transmission lines froze heavily and many of them collapsed completely under the excessive weight—resulting in a month of powerlessness in a huge but somewhat less densely populated area.

Conclusion

In the context of environmental disasters and election integrity problems, the preceding analysis suggests that much more attention needs to be devoted in the future to proactive planning for adversities, rather than simply waiting for the next environmental catastrophe, or the next heatedly disputed local or national election. Much greater accountability, contingency planning, and objective oversight are needed—along with considerably greater non-partisan even-handedness—to ensure that the effects of future environmental disasters can be less widespread and that future elections will be able to avoid problems that are likely to recur or unfold anew in the future. Thus, it seems that a common link between election integrity and environmental emergency preparedness lies in increased understanding of the risks and greater foresight in anticipating what can go wrong. ■

Peter G. Neumann (neumann@csl.sri.com), Principal Scientist in SRI International's Computer Science Laboratory, is moderator of the ACM Risks Forum (<http://www.risks.org>). Please check out recent issues of RISKS for source information and see his website for further background (<http://www.csl.sri.com/neumann>).

Copyright held by author.



George V. Neville-Neil

DOI:10.1145/2408776.2408786

Article development led by **acmqueue**
queue.acm.org

Kode Vicious Divided by Division

Is there a “best used by” date for software?

Dear KV,

Do you know of any rule of thumb for how often a piece of software should need maintenance? I am not thinking about bug fixes, since bugs are there from the moment the code is written, but about the constant refactoring that seems to go on in code. Sometimes I feel as if programmers use refactoring as a way of keeping their jobs, rather than offering any real improvement. Is there a “best used by” date for software?

Fresh Code

Dear Fresh,

I definitely like the idea of software coming with a freshness mark like you see on milk cartons. As with other perishable products, software does seem to go bad after a while, and I am often reminded of the stench of spoiled milk when I open certain source files in my editor. No wonder so many programmers grimace whenever they go to fix bugs.

I think that a better analogy for software is that of infrastructure. Anyone who has lived long enough to see new infrastructure built, neglected, and then repaired should understand this analogy. Consider the highways built in the U.S. in the 1950s. When these roads were first built they were considered a marvel, helping commuters get into and out of large cities. Everyone loves something new, and the building of this infrastructure was heralded with a good deal of fanfare, speeches, and other celebratory events that you associate with large projects.

Once completed, however, the process of neglect sets in—cost cutting, slow repairs, ignoring major design flaws until bits of the roadways fall down. Finally, the highway is so poorly maintained that it is a menace, and then, unless you get lucky and an earthquake destroys the hideous thing, you come to the usual engineering decision: repair or rebuild.

The difference with software is that if code is used in the same way, day in and day out, and never extended or changed—other than fixing previously existing bugs—it should not wear out. Not wearing out depends on a few things—especially that hardware does not advance. A working system delivered in 1980—on, say, a classic mini-computer such as the VAX—should, if the same hardware is present, work the same today as it did when it was built.

The problems of software maintenance arise because things change.

While the original libraries used to build a system do not wear out in any physical sense, the code they interact with changes over time as idiots (oops, I meant to say marketers) demand new features and as the speed and complexity of hardware advances. Efforts at portability are noble and often worthwhile, but there is simply no way that a piece of code that ran on a 1-MIPS CISC (complex instruction set computing) computer is going to run—without significant retesting and changes—on a modern processor with modern peripherals. Operating systems and device drivers can go only so far to hide the underlying changes from applications.

While I have seen plenty of navel-gazing exercises masquerading as refactoring, there comes a time in the life of all software when the design decisions it expresses must be reexamined. There is no hard and fast limit for this. If the code was a “prototype”—you know, code that management swore up and down they would never use, and then did—it is going to go bad sooner rather than later.

Programs that were written in a more reasonable style and without ridiculous schedules imposed from above maintain their freshness longer. I consider my own code to have a “best by date” of one year from when I complete the project. If I have not looked at some code in a year, I have probably forgotten how it worked, anyway.

KV





An APL keyboard.

Dear KV,

I have been upgrading some Python 2 code to Python 3 and ran across the following change in the language. It used to be that division (/) of two integers resulted in an integer, but to get that functionality in Python 3, I need to use //. There is still a /, but that is different. Why would anyone in their right mind have two similar operations that are so closely coded? Don't they know this will lead to errors?

Divided by Division

Dear Divided,

Python is not the first—and I am quite sure it will not be the last—language to use visually similar keywords to mean different things. Consider C and C++ where bitwise and logical operations use very similar images to mean totally different operations: | for bitwise or operation and || for the logical, for example. I also recently discovered this change in Python 3 and my coworkers discovered it just after I did, as I was quite vocal in my reaction.

The problem of not having visually distinctive images in programming goes back to the problem, alluded to by Poul-Henning Kamp (“Sir, Please Step Away from the ASR-33!,” *Communications*, November 2010), of the character set we use to create our languages. Language designers have only the character set shown in the accompanying figure to work with when they are looking for something to represent a shortcut to an operation. Many of the characters already have well-established meanings outside

Language character set.

```
!"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~
```

of programming, such as the arithmetic operations +, -, *, and /, and the language designer who decides to change their meanings should be severely punished.

It is certainly possible to forgo shortcuts and to make everything a function such as (plus a b) for functional syntax, or create a large list of reserved words as in a equals b plus c for Algol-like languages. The fact is, as programmers, we like compact syntax and would balk at using something as bulky as the example I have just given.

Another alternative is to throw away ASCII encoding and move to something richer in which we can have more distinct images to which we can attach semantic meanings. The problem then arises of how to type in that code. Modern computer keyboards are meant to allow programmers to type ASCII. Ask Japanese programmers whether they use a Japanese keyboard or an American one, and nine out of 10 will tell you an American one. They choose the U.S. version because the “programmer keys,” the ones that represent the glyphs shown in the figure, are in the easiest-to-use placement. Extending our character set to allow for complex glyphs will slow the process of entering new code, and we all know that typing speed is the biggest indicator of code quality. Many years

ago there was a language called APL that required a special keyboard. That language is mostly dead—look at the keyboard shown here to find out why.

That brings us to where we are now with / meaning one thing and // meaning another. I am quite sure many bugs will result from this conflation of images, and I am sure they are going to occur when the person working on the code has just been awakened from a deep sleep by a panicked telephone call. In the light of day, it is easy to tell / from //, but in the dim light of reawakening, it is not so easy.

KV

Related articles on queue.acm.org

You Don't Know Jack About Software Maintenance

Paul Stachour, David Collier-Brown
<http://queue.acm.org/detail.cfm?id=1640399>

The Meaning of Maintenance

George V. Neville-Neil
<http://queue.acm.org/detail.cfm?id=1594861>

The Software Industry IS the Problem

Poul-Henning Kamp
<http://queue.acm.org/detail.cfm?id=2030258>

Concurrency's Shysters

January 14, 2009
http://blogs.sun.com/bmc/entry/concurrency_s_shysters

George V. Neville-Neil (kv@v.acm.org) is the proprietor of Neville-Neil Consulting and a member 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.

Copyright held by author.

Education

Reflections on Stanford's MOOCs

New possibilities in online education create new challenges.

THE RECENT WAVE of Massive Open Online Courses (MOOCs) has highlighted the potential for making educational offerings accessible at a global level. The attention MOOCs have received is well deserved, but it belies the fact that various forms of online education have existed for many years. Rather than attempting to catalogue the broad spectrum of online learning resources, we focus on a sampling of initiatives in online education—with an emphasis on our home institution, Stanford University, with which we are most familiar—highlighting some of the opportunities and challenges at hand.

By way of background, the Stanford Center for Professional Development (SCPD) began offering distance-learning courses via television microwave channels in 1969. Students—mostly engineers working in the local Silicon Valley area—had the opportunity to watch course lectures on television at off-campus locations and submit course work via a courier system. By 1996, these course offerings had evolved to using streaming video via the Internet and a variety of means for electronic assignment submission and distribution of course materials. While in some respects these course offerings were similar to the more modern MOOCs, a critical differentiator was cost. Students enrolled in SCPD courses paid tuition



The Stanford Engineering Everywhere website (<http://see.stanford.edu>) was launched in 2008.

akin to on-campus Stanford students, received the same level of service in the evaluation of their work, and received course credit upon passing classes. The costs involved, however, meant the number of remote students enrolled in the Stanford courses was usually dwarfed by the number of on-campus students in the courses.

That equation changed in 2008 with the launch of Stanford Engineering

Everywhere (SEE; <http://see.stanford.edu>). Through the SEE initiative, Stanford made several of its most popular engineering courses—including six CS courses—freely available online, including full videos of class lectures and all course materials (handouts, assignments, and software). Course videos were released through YouTube, Apple's iTunes University, and on Stanford's own website. While

the SEE initiative was not itself novel (MIT's OpenCourseWare project was created years prior to SEE), the response to the open online materials was significant. Among the classes released was an offering of CS106A (Stanford's Java-based CS1 course). That course's (hour-long) lecture videos have been viewed more than *two million* times on YouTube alone. The availability of such online materials has resulted in universities in the U.S., China, India, and Brazil using these videos as part of teaching their own introductory programming courses. And since the CS106A course covers much of the same material as the AP computer science curriculum, students without access to traditional CS courses at high school have reported being able to study for and pass the AP CS exam as a result of watching the course videos (either alone or through a teacher-guided independent study at their school). Of note is the fact that these online materials generated such a strong response without any of the affordances of MOOCs (which typically offer enrollment, quizzes and assessments, assignment deadlines, statements of accomplishment, and so forth). Importantly, it is these seemingly evolutionary additional features that allowed the recent set of MOOCs to cross a line from being considered yet another free educational *resource* to being viewed as scalable free *courses*. This change in perception also brought with it a new set of possibilities and expectations.

The trio of MOOCs released by Stanford faculty in fall 2011—courses in artificial intelligence, databases, and machine learning—attracted hundreds of thousands of students and spawned two private ventures: Coursera and Udacity. Concurrently, edX evolved from MITx as a non-profit consortium for online education, comprised initially of MIT and Harvard, with UC Berkeley and the University of Texas later joining forces. Stanford has also developed two new online learning platforms—Class2Go and Venture Lab—and committed itself to further work in this area by appointing CS professor John Mitchell as the inaugural Vice Provost for Online Education.

MOOCs have the potential to pro-

Many challenges remain if MOOCs are to become competitive with the “classical” model of in-class education.

vide education on a global scale. But many challenges remain if MOOCs, either in a standalone or hybrid context, are to become competitive with the “classical” model of in-class education. Here, we discuss some of the opportunities and challenges facing MOOCs based on our experiences.

Validation and Plagiarism

Perhaps the most widely discussed challenge in online education is that of validating original work and preventing (or at least detecting) plagiarism. It has been reported that plagiarism is a potentially significant problem in online courses.¹⁰ In response, Coursera has stated it may attempt to employ plagiarism-detection software. It is too early to tell the efficacy of automated methods for plagiarism detection, but the clear need may motivate further research in this area. Both edX and Udacity have partnered with Pearson VUE, a provider of testing centers, to validate students taking proctored exams.^{4,7} While the use of testing centers to validate students' identity and original work seems more straightforward in practice than automated methods for plagiarism detection, it also carries with it cost for the student. How such costs are to be weighed with respect to the costs and benefits of enrolling in a traditional course will be an important factor in the future success of MOOCs.

Certification

Another important component of MOOCs is whether and how they

provide some form of certification to students. While the experience with SEE (which provides no form of certification) leads us to believe that many students will still pursue online education regardless of certification, we recognize that many students will be concerned they receive external validation/certification of their learning. There are many companies (both non-profit and for-profit) that have experience in awarding various forms of certifications, and several colleges offer purely online certificates and even degrees. Hybrid models are also emerging. For example, the University of Washington has offered to give college credit for some of its courses taken through Coursera for students who pay a fee and complete additional assessments.² Thus, models for the certification of online work certainly exist. The extent to which such certifications are recognized by others, especially employers, will certainly impact how MOOCs are viewed relative to more traditional courses.

Richer Evaluation

Although the initial set of MOOCs focused on relatively straightforward means for evaluation, such as multiple-choice quizzes or short-answer questions, richer evaluation models measuring student engagement more fully with the material soon emerged. In a computing context, such evaluations include assessing students' programs and assessing larger student projects. While mechanisms such as testing suites can be used to measure aspects of a program's functionality, such tests are not applicable in all contexts, such as with interactive applications. Care must be given in developing assessment platforms allowing for a rich design space of assignments while still making (semi-) automated assessment feasible.

Peer assessments have also been proposed as a means for providing human assessments at scale. Research in peer assessment has shown the potential for this approach.⁶ For example, Stanford's online Human-Computer Interaction course, taught by Scott Klemmer, emphasizes student design work. Enabling assessment of these designs at scale requires each student in the course to provide an or-

dering of several designs with which they are presented. The orderings from all students are then combined to get a more global ranking of designs. Interestingly, some of the designs presented to students to order have been graded by human experts (for example, the teaching assistants for the on-campus course). Since these graded designs are now embedded in the global ordering, a grade can be determined for a student-submitted design by determining how it ranks relative to expert-evaluated designs. While such a system is not without issues, it does provide an interesting model for injecting expert evaluation into a primarily peer-based assessment scheme.

Personalized Education

The massive scale of MOOCs provides the opportunity to collect unprecedented volumes of data on students' interactions with learning systems. As a result, it becomes possible to use machine learning to gain insight on and potentially personalize human learning. Work in this vein has existed for years under the rubric of intelligent tutoring systems and educational data mining. As one recent example, Piech et al.⁵ applied machine learning techniques to build probabilistic models of automatically logged intermediate versions of student programs in our CS106A course. Such models, built on initial assignments in the course, were better predictors of students' performance later in the course than the grades on those assignments. Such techniques could be used to identify students who are struggling in an online course and suggest remediation via alternative learning paths through a MOOC.

MOOCs also have the potential to present information to students using many different pedagogic approaches, allowing each student to select a particular desired approach, or even making such suggestions to the student. A meta-analysis of more than 1,000 online studies⁸ argues that features such as instructor-directed and collaborative online instruction led to improved learning for students, and that blended learning environments tended to be better for

We need to identify new ways to think about online learning.

students than purely online ones. Online courses can evolve to incorporate such identified best practices.

Hybrid Education

As evidenced by Martin,³ some universities are leveraging MOOCs by having their students watch videos from an online course—Stanford's artificial intelligence class in Martin's case—prior to attending class at their own university to discuss the material and engage in additional assessments. Such “flipped classrooms,” which existed in various forms before MOOCs, enable the instructor to spend less time lecturing and more time interacting with the students. Indeed, we are likely only scratching the surface in exploring ways in which online videos can augment or potentially improve education. More work is needed to determine what instruction students should do on their own in preparing for class, as well as identifying how best to utilize class time given the fact that students have watched videos and engaged in attendant exercises in preparation.

Innovation Beyond Online Videos

We need to identify new ways to think about online learning. Tools, such as algorithm visualizations (for example, AlgoViz, <http://algoviz.org>, or Amit Patel's probability visualizations, <http://www.redblobgames.com>), programming practice environments (such as Nick Parlante's CodingBat, <http://codingbat.com>, or Amruth Kumar's Problets, <http://problets.org>), and editable coding visualizers (such as Philip Guo's Online Python Tutor, <http://www.pythontutor.com>) all offer promising online environments to aid student learning. We believe such innovations can become especially effective in online education, augment-

ing video presentations with myriad interactive activities for the learner to perform. Perhaps incorporation of appropriate interactive aids can begin to move closer toward identifying and constructing curricula for making Alan Kay's Dynabook¹ a reality.

It was Thomas Edison who believed that the advent of the phonograph would completely revolutionize education, rendering teachers obsolete. In the intervening century, similar predictions have been made about many other technological innovations. We do not believe MOOCs are going to render teachers obsolete, certainly not in the foreseeable future. Online education can augment more traditional instruction, and serve as an effective means to scale education to students when other (in-person) forms of instruction are unavailable. Like Vardi,⁹ we believe MOOCs are here to stay. However, we are much more positive about online education's transformative potential, if we as a community can find solutions to the challenges at hand. It is really up to us. ■

References

1. Kay, A. A personal computer for children of all ages. In *Proceedings of the ACM Annual Conference—Volume 1* (1972).
2. Long, K. UW to offer fee-based courses through Coursera. *The Seattle Times* (July 18, 2012).
3. Martin, F. Will massive open online courses change how we teach? *Commun. ACM* 55, 8 (Aug. 2012), 26–28.
4. Parry, M. EdX offers proctored exams for open online course. *The Chronicle of Higher Education* (Sept. 6, 2012).
5. Piech, C., Sahami, M., Koller, D., Cooper, S. and Blikstein, P. Modeling how students learn to program. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education (SIGCSE '12)*. ACM, New York, 2012, 153–160.
6. Sadlar, P. and Good, E. The impact of self- and peer-grading on student learning. *Educational Assessment* 11, 1 (2006), 1–31.
7. Udacity blog. Udacity in partnership with Pearson VUE announces testing centers. (June 1, 2012); <http://blog.udacity.com/2012/06/udacity-in-partnership-with-pearson-vue.html>.
8. U.S. Department of Education, Office of Planning, Evaluation, and Policy Development. Evaluation of evidence-based practices in online learning: A meta-analysis and review of online learning studies. Washington, D.C., 2010.
9. Vardi, M.Y. Will MOOCs destroy academia? *Commun. ACM* 55, 11 (Nov. 2012), 5.
10. Young, J. Dozens of plagiarism incidents are reported in Coursera's free online courses. *The Chronicle of Higher Education* (Aug. 16, 2012).

Steve Cooper (coopers@cs.stanford.edu) is an associate professor in the computer science department at Stanford University.

Mehran Sahami (sahami@cs.stanford.edu) is an associate professor in the computer science department at Stanford University.

Copyright held by author.

Economic and Business Dimensions

The Value of Microprocessor Designs

Applying a centuries-old technique to modern cost estimation.

OVER THE YEARS the leading microprocessor company, Intel, has introduced a steady stream of new microprocessor designs: the 286, 386, 486, Pentium, Pentium II, Pentium III, Pentium 4, and more recently the Multicore design. In the microprocessor industry these designs are called *microarchitectures*. If there was a market for microarchitectures what would each design sell for? Our research addresses that modern question using economic insights developed almost two centuries ago.

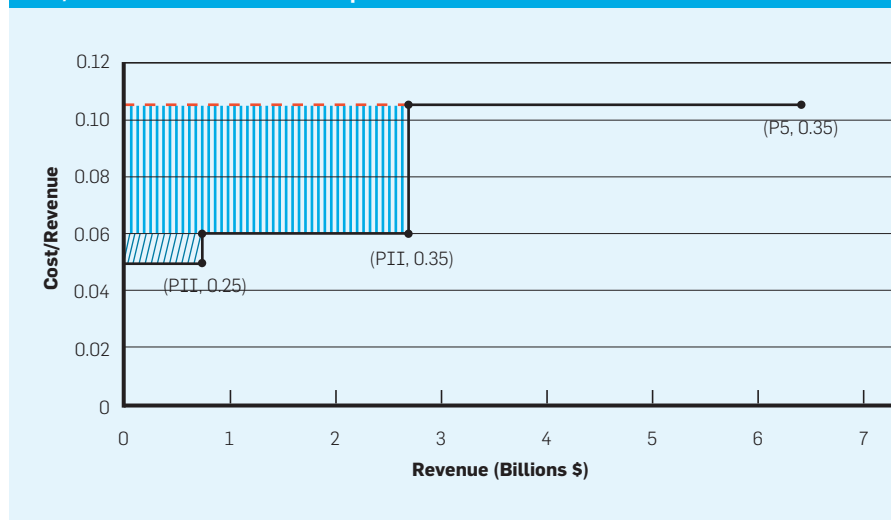
Why estimate the value of these designs? This type of calculation can inform many aspects of firm strategy and valuation. Many companies develop new designs and increasingly outsource the manufacturing of the products. For example, fabless companies like Qualcomm and Broadcom design chips and do none of their own manufacturing. Foundries do it for them on contract. The value of the fabless firms depends predominantly on the value of designs.

Many factors complicate any attempt to estimate the value of intellectual property associated with such product designs. In microprocessors for instance, consumers are not willing to pay for a new design per se, but for the increase in computing power that comes with a new design.

The classical economist, David Ricardo, had the key insight in 1817.² Ricardo asked how much would be the rent to a unit of fertile land (say close to a river). Ricardo reasoned that producing a pound of corn in the fertile land is less costly to a farmer than producing a pound on marginal land (the worst land being cultivated, say in the hills). The fertile land requires less effort to achieve the same output. The rent for the fertile land arises from the difference between the labor cost of producing the same quantity of crop on the fertile land and on the marginal land. Charge the farmer a rent higher than this maximum, and the farmer would prefer to move out and start cultivating the marginal land.

Ricardo's logic still applies today, and can help estimate the rent to a new microprocessor design at any point in time. Think of a new design as analogous to fertile land and an old design as the marginal land. A microprocessor made with a new design can compute faster and hence sells for a higher price. To get the same revenue from selling microprocessors with older designs, Intel would need to sell more of the older microprocessors, something that involves more labor cost than making one microprocessor with the new design. The rent for the new design, therefore, is the difference between the cost of producing a dollar of revenue with the new design and the cost using the old

Cost/revenue ratios for three Intel processors.



Estimate of the total value of the Intel design or process technology.

Design	Cost Savings*	Process Technology	Cost Savings*
486	0.26	0.80	0.96
Pentium	2.74	0.60	1.18
Pentium II	0.82	0.35	5.16
Pentium III	4.30	0.25	1.64
Pentium-M	1.35	0.18	3.00
Pentium 4	0.01	0.13	14.40
CORE	2.63	0.09	5.01
		0.065	2.08
TOTAL	11.85	TOTAL	33.43

*Cost savings in billions of dollars.

design in a given time period. Adding up all such rent across the lifetime of a specific design provides an estimate of the value of the design to Intel.

It is not quite that simple in practice, of course. Increases in computing power can come from factors other than a new design. The decreasing size of transistors used in microprocessors is the leading example. Advances in semiconductor process technology have steadily driven down transistor sizes from three microns (where a micron is a millionth of a meter) in the original 8086 made by Intel to around 0.022 microns in the latest chip. These smaller transistors—by themselves—lead to greater computing power, without any improvements in designs. In other words, a proper estimate must separate the value provided to Intel by new designs from the value provided by the technological transitions to smaller transistors.

To summarize, Ricardo's logic still applies. It can be applied to the measurement of rent to a new microprocessor design at any point in time. A new microprocessor can be defined as a pair of attributes: the design it uses and the semiconductor process technology it was made with. The rent to the combination of design and process technology used in a new microprocessor is the cost savings the new microprocessor provides over the oldest one currently in use. If the new microprocessor uses the same process technology as the oldest one currently in use, then the difference in the cost of production is the rent to the design. On the other hand, if the new microprocessor uses the same design as the oldest one currently in use, the differ-

ence in cost of production is the rent to the process technology.

The accompanying figure illustrates the approach, using data from Intel's production for the third quarter of 1997. There were two designs in operation in the quarter, Pentium (P5) and Pentium II (PII). There were process technologies, 0.35 microns and 0.25 microns. There were three microprocessor vintages in production, (P5, 0.35), (PII, 0.35), and (PII, 0.25). The y-axis shows the average cost/average price of microprocessors of each vintage. The x-axis shows the total revenue obtained from each vintage.

What is the message of the figure? The (P5, 0.35) vintage of microprocessors were the oldest ones, and had a high cost/revenue ratio. Microprocessors produced with the same process technology, but with a new design (PII) had a lower cost/price ratio. The latest microprocessors featuring a new design (PII) and a new process technology (0.25) had the lowest cost/price ratio among the three vintages. The vertical distance between the top and the second horizontal lines is the cost saving provided by the new design PII over the old design P5, for each dollar of revenue that Intel obtained by selling the chips that used PII design. Hence the total rent to PII design during the quarter is the area of the rectangle shaded with vertical lines. Similarly, the area of the rectangle shaded with diagonal lines is the rent to 0.25-micron process technology during the quarter.

An estimate of the total value of the design or process technology to Intel comes from constructing similar diagrams for all the quarters in which a design or process technology was in use and by adding these up. The ac-

companying table shows the estimates we obtained in our study.¹

The cost savings from new design is in many billions of U.S. dollars. One can see from the table that Intel's savings from new process technologies was almost three times the savings from new microprocessor designs, indicating the relative importance of new manufacturing technology transitions to Intel.

The table also includes information about the value of specific designs. The Pentium 4 design provided very little value to Intel. This is not surprising due to its problems with overheating, which forced Intel to move to the new multicore designs. The Pentium III was the most valuable design for Intel, reflecting perhaps the high price that new designs were able to command at the height of the Internet boom of the late 1990s. In that period, Intel used the 0.13 manufacturing technology, again the one that we estimated to have provided most value to Intel.

It should be noted that these values are calculated ex-post, after the microprocessors that used these designs were sold on the market. This method is not appropriate for forecasting value prior to any market experience, an important precaution in interpreting these figures. Intel probably spent the most on developing the Pentium 4 among all its designs, an investment that did not pan out for Intel.

Overall, these calculations provide a rough estimate of the value to Intel of intellectual property embedded in new designs. Ricardo's centuries-old wisdom on land rents turns out to be insightful for valuing intangible assets. **■**

References

1. Aizcorbe, A., Kortum, S., and Pillai, U. Measuring Knowledge Capital and its Contribution to Productivity Growth. 2012, Working paper.
2. Ricardo, D. *Principles of Political Economy and Taxation*. London, U.K., 1817.

Ana Aizcorbe (ana.aizcorbe@bea.gov) is Chief Economist, Bureau of Economic Analysis, U.S. Department of Commerce in Washington, D.C.

Samuel Kortum (samuel.kortum@yale.edu) is a professor in the Department of Economics at Yale University in New Haven, CT.

Unni Pillai (usadasivanpillai@albany.edu) is an assistant professor in the College of Nanoscale Science and Engineering at the State University of New York at Albany.

The opinions expressed in this column are those of the authors and do not represent those of the U.S. Bureau of Economic Analysis.

Copyright held by author.

Viewpoint Cloud Services Certification

How to address the lack of transparency, trust, and acceptance in cloud services.

CLOUD COMPUTING IS AN evolving paradigm that affects a large part of the IT industry, in particular the way hardware and software are deployed: as a service.¹ Cloud computing provides new opportunities for IT service providers, such as the adoption of new business models and the realization of economies of scale by increasing efficiency of resource utilization. Adopters are supposed to benefit from advantages like up-to-date IT resources with a high degree of flexibility and low upfront capital investments.⁶

However, despite advantages of cloud computing, small and medium enterprises (SMEs) in particular remain cautious implementing cloud service solutions.⁴ This holds true for both IT service providers and IT service users. The main reasons for the reluctance of companies to adopt cloud computing include:

- ▶ Due to the prevailing information asymmetry on the market, companies have difficulties comprehensively assessing the individual benefits and challenges associated with the adoption of cloud services. Furthermore, the information asymmetry impedes providers from aligning their services with the needs of potential customers.

- ▶ Companies lack appropriate, qualified, trustworthy information and benchmarks to assess cloud services with regard to individual benefits and associated risks.



- ▶ Companies lack approaches and metrics to adequately assess and compare the service quality of cloud services, especially, regarding security and reliability.

- ▶ Industry-specific requirements and restrictions on IT usage and data processing limit the adoption of cloud services in sectors like health care or banking. Many of those requirements and restrictions are outdated and were issued long before broadband Internet connections and mobile devices became ubiquitous.

- ▶ Noteworthy uncertainties concerning legal compliance and conformance with international privacy requirements can be observed. Providers are constantly faced with the challenge to design niche-oriented, demand-specific services in a legally compliant manner.

Reflecting these reasons for inhibiting cloud computing adoption, the environment surrounding cloud computing is characterized by uncertainty and a lack of transparency. Yet, trust is necessary in situations in which the interested party is confronted with un-

certainity.⁷ Addressing the present trust issues in cloud computing and promoting transparent information exchange between cloud service providers and cloud users are essential premises to accomplish broad diffusion of cloud computing in the market.

Certification of Cloud Services

We believe certification of cloud services by independent certification institutions can cope with the challenging lack of transparency, trust, and acceptance. Research has shown that trust can be built through supporting IT-based mechanisms like certifications and escrows if experience in a market is not readily available.⁸ Furthermore, certifications help to establish market transparency, which companies may not be able to achieve on their own. Potential cloud adopters are faced with an abundance of service offerings of similar functionality. SMEs may not have sufficient resources to adequately assess cloud services, whereas large enterprises may have the resources, but still have to raise funds and undergo significant efforts in order to assess and benchmark cloud services. Ultimately, all companies, which are planning to adopt a cloud service, need to perform similar assessments. Thus, it is economically beneficial to dedicate these assessments to specialized organizations, which issue broadly accepted and standardized certifications.

A cloud service certification process should include on-site data center audits as well as extensive evaluation of contracts and services. In order to achieve such a certification, a cloud service should satisfy specific quality specifications including contractual requirements (for example, service level agreements), legal requirements (for example, privacy policy), security requirements (for example, encryption), functional requirements (for example, API implementations), business processes (for example, quality management), and data center infrastructure (for example, physical access control). Additional industry-specific requirements may apply (for example, in the domain of health care^{3,9}). In fact, models to assess perceived quality of service have been extensively researched and validated in practice.⁵ Assessments of service quality by a

provider's customers or independent third parties can improve trust and acceptance of a service. This approach has successfully been applied in other service industries, for example, sale of goods, news media, or entertainment.²

Reflecting the aforementioned reasons for adoption uncertainty, a certification is particularly beneficial in the following scenarios:

Security and trust. The implementation of cloud computing creates additional challenges concerning IT security. Besides technical issues, customers need to trust in the security and reliability of a service in order to adopt it. In the case of online banking or online shopping, public key certificates issued by certificate authorities are a common way to verify a website's authenticity and promote customers' trust. Extended Validation Certificates do not differ in structure or cryptography from other (cheaper) certificates, but require extensive identity verification of the requesting organization. Thus, the online transaction itself is not more secure (according to its encryption), but the certification is presented more prominently to the user and the extended validation fosters the trustfulness of the website. In the context of cloud computing, a certification by an independent certification authority can improve trust the same way as in the domains of online banking and online shopping. In addition to the provider's identity, a cloud certificate could evaluate infrastructure security and IT security measures of the cloud service provider. We consider the certification of large infrastructure, platform, or software providers as important since these providers serve as hubs for enormous amounts of data.

Certifications help to establish market transparency, which companies may not be able to achieve on their own.

Therefore, security flaws or outages in the systems of these large providers affect a vast number of cloud users.

Legal compliance and privacy. Current discussions on legal conflicts between the United States Patriot Act and the European Union (EU) Data Protection Directive (95/46/EC) intensify the need for legally compliant cloud services. Moreover, individual member states of the EU have implemented the 16-year-old EU data protection directive in very different manners. As a consequence, cloud service providers must deal thoroughly with 27 different policies in order to comply with all 27 EU member states' data protection laws. In addition, sector-specific regulations may apply (for example, the Health Insurance Portability and Accountability Act in the U.S.). Implementing a framework with clear guidelines for privacy and legal compliance of cloud services would support providers to design and implement compliant cloud solutions. Cloud service certifications verifying the adherence to such a legal and privacy framework can support users in their adoption decisions as they can rely on the ongoing legal compliance of certified cloud services. Likewise, specialized cloud service providers can benefit from cloud certifications when selecting platform or infrastructure providers to deploy their services, which need to adhere to the national or industry-specific requirements of their customers.

Digital preservation and lock-in effects. Digital preservation describes the management of digital information in order to keep it accessible, reproducible, and interpretable over long periods of time and different innovation cycles. Digital preservation does not only focus on preserving data, but also on preserving the representation information necessary to interpret the preserved data. For example, the representation information may be an application used to access and interpret the data or specifications of the data format. In cloud computing, hardware and software are delivered as a service and are not in possession of the user. Thus, neither data nor applications are physically accessible. Moreover, data formats in cloud services like Google Docs are opaque. Supporting digital preservation of cloud-based informa-

tion and applications might be included in the certification requirements for cloud services. Another challenge for cloud service providers includes the prevention of lock-in effects. In order to acquire a certification, interfaces for digital preservation and data migration to other cloud service providers need to be provided.

Transparency. As a result of the late-2000s financial crisis, customers lost their confidence in the banking industry. Risky, complex, and non-transparent financial products, such as mortgage-backed securities or collateralized debt obligations, were placed on the capital market as supposedly secure investments. Applying this situation correspondingly on cloud services, users do not necessarily know which cloud services they are actually using and where data will be processed and stored. A Software as a Service provider in Germany may provide a cloud service, which integrates the capabilities of several cloud services in Europe, Asia, and North America. The provider may implement the service within a Platform as a Service environment in the U.S., which in turn utilizes databases at an Infrastructure as a Service provider in Ireland and sources computing power from a cloud marketplace like Spotcloud (a marketplace for cloud service providers to sell their unused cloud capacity). Cloud adopters will contract and interact with a German provider, assuming the strict German privacy restrictions apply, but in fact it is totally opaque where data is processed and stored. But the concept of cloud computing does not need to be cloudy at all. The clarification of a service's interrelations as part of the certification requirements can clarify complex provider cooperation and interaction.

Challenges for Cloud Certifications

Cloud service certifications can resolve adoption uncertainties and thereby support users and providers of cloud services in their adoption decisions. However, adherence to certification standards also entails challenges that need to be considered:

► Particularly, small- and medium-sized cloud service providers may not have the budget to acquire a certification for their cloud services; therefore,

We believe introducing a certification for cloud services is a step forward to a more trustworthy and transparent cloud computing environment.

they would have to struggle with a competitive disadvantage. A certification needs to be affordable, but nevertheless comprehensive in terms of on-site auditing and contractual evaluations.

► The demands of maintaining certifications may preclude small cloud service providers from delivering services in a cost-effective manner, while large cloud service providers can continue to differentiate themselves by their ability to provide significant cost savings and a high level of resource elasticity to their customers. Thus, large cloud service providers can neglect undergoing audits of their physical facilities, services, and processes and accomplish a similar outcome to certification by solely relying on their reputation. In contrast, small cloud service providers may be urged to undergo certification audits in order to differentiate themselves on the market and thereby suffer a competitive disadvantage.

► Certifications need to balance the tension between usefulness and complexity. A certification framework may slow down innovation if adherence to the framework is connected with very strict requirements. But innovative, pioneering services, and short innovation cycles are main benefits of the cloud computing paradigm. Therefore, a certification framework needs to be flexible and adaptable in order to cope with the fast innovation cycles of the IT industry. However, due to the wide diversity of cloud service offerings, designing a comprehensive and widely applicable certification framework includes the risk of devolving into a set of lowest com-

Calendar of Events

February 18–20

Third ACM Conference on Data and Application Security and Privacy, San Antonio, TX, Contact: Elisa Bertino, Phone: 765-496-2399, Email: bertino@cerias.purdue.edu

February 23–27

Computer Supported Cooperative Work, Computer History Museum of Silicon Valley, Contact: Magali Cohen, Phone: 650-964-2858, Email: magaliis@pmwintl.com

February 23–27

ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, Shenzhen, China, Sponsored: SIGPLAN, Contact: Alexandru Nicolau, Phone: 949-824-4079, Email: nicolau@ics.uci.edu

February 26–27

14th Workshop on Mobile Computing Systems and Applications, Florida, Sponsored: SIGMOBILE, Contact: Sharad Agarwal, Phone: 425-722-5521, Email: sharad.agarwal@microsoft.com

February 27–March 1

International Symposium on Engineering Secure Software and Systems, Rocquencourt, France, Contact: Valerie Issarny, Email: Valerie.issarny@inria.fr

February 27–March 1

Multimedia Systems Conference 2013, Oslo, Norway, Sponsored: SIGMM, Contact: Carsten Griwodz, Email: griff+acm@ifi.uio.no

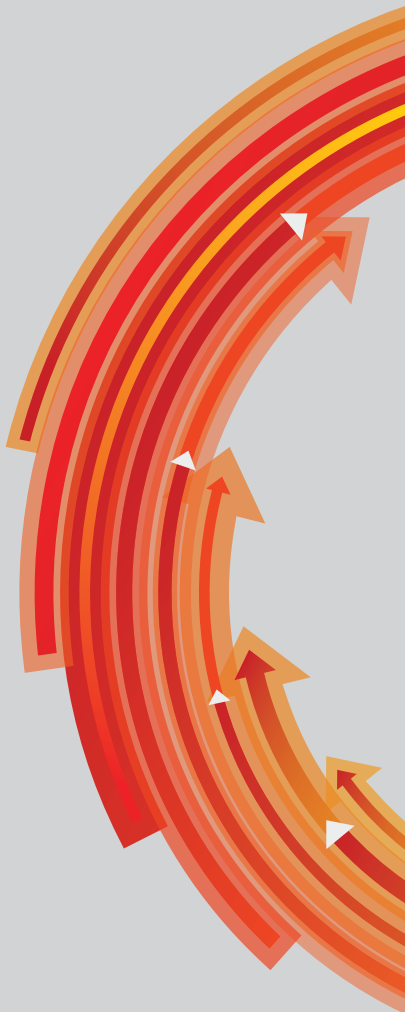
March

March 3–6

ACM/IEEE International Conference on Human-Robot Interaction, Tokyo, Japan, Sponsored: SIGART and SIGCHI, Contact: Hideaki Kuzuoka, Email: kuzuoka@iit.tsukuba.ac.jp

Computing Reviews is on the move

Our new URL is
ComputingReviews.com



COMPUTING REVIEWS

A daily snapshot of what is new
and hot in computing.

mon denominator standards, which in turn would undermine the desired outcomes of a certification.


► Trustworthy certification institutions need to be appointed in order to ensure acceptance of the certification. Decision makers of cloud service providers and cloud adopters must trust the certification authority; otherwise, the credibility of certified cloud services would be undermined.

► Existing cloud service certifications are valid for a predefined time-frame (usually two years) and only provide a snapshot of the situation before and during the time of the audit. Whether the certified criteria are met during the validity period cannot be ensured. By implementing automated certification processes for continuous monitoring and refreshment of the certification in addition to periodic on-site audits (for example, biennially), a constant level of service quality can be monitored and proved, which regular on-site audits cannot accomplish in an economic manner.

Conclusion

Considering the current situation on the cloud computing market, unresolved obstacles need to be addressed for effective development and diffusion of innovative cloud services. A standardized certification for cloud services aims to establish trust and improves acceptance of the cloud computing paradigm. Small, medium, and large cloud service providers as well as cloud users can benefit from the outcomes of established cloud service certifications. By achieving practice-oriented and market-relevant certificates for their cloud services, small and regionally oriented IT service providers can stand out in the marketplace and gain a broader customer base. Furthermore, mid-sized IT service providers can implement legally compliant, customer-specific requirements, which cannot be satisfied by usually highly standardized solutions of large service providers. By signaling valuable qualities like transparency of their services, legal compliance, reliable service levels, and a high level of security at their data centers, large providers can attract other cloud service providers to utilize their services instead of main-

taining similar services in-house. By producing trustworthy cloud service certifications, cloud adopters are able to identify risks and benefits of individual cloud services and consider those in their adoption decisions.

Currently, organizations such as Cloud Security Alliance and EuroCloud are launching cloud certification programs for individuals, providers, or services. We emphasize the need for broadly accepted, established, and feasible cloud service certification solutions as well as trustworthy auditing institutions. Time will tell if certifications can mitigate challenges concerning transparency, trust, and acceptance and whether current providers can cope with the outlined challenges of a certification itself. We want to motivate researchers and practitioners to engage in topics concerning cloud service certifications. We believe introducing a certification for cloud services is one possible way to address the current gaps and issues in cloud computing, and that it is a step forward to a more trustworthy and transparent cloud computing environment. 

References

1. Armbrust, M. et al. A view of cloud computing. *Commun. ACM* 53, 4 (Apr. 2010), 50–58.
2. Dellarocas, C. The digitization of word of mouth: Promise and challenges of online feedback mechanisms. *Management Science* 10, (2003) 1407–1424.
3. Dünnebeil, S. et al. Determinants of physicians' technology acceptance for e-health in ambulatory care. *International Journal of Medical Informatics*, 2012; DOI: 10.1016/j.ijmedinf.2012.02.002.
4. European Commission, Cloud Computing: Public Consultation Report, European Commission, 2011; http://ec.europa.eu/information_society/activities/cloudcomputing/docs/ccconsultationfinalreport.pdf.
5. Fassnacht, M. and Koese, I. Quality of electronic services. *Journal of Service Research* 9, 1 (2006), 19–37.
6. Marston, S. et al. Cloud computing—The business perspective. *Decision Support Systems* 51, 1 (2011), 176–189.
7. Mayer, R.C., Davis, J.H., and Schoorman, F.D. An integrative model of organizational trust. *Academy of Management Review* 20, 3 (1995), 709–734.
8. Pavlou, P. and Gefen, D. Building effective online marketplaces with institution-based trust. *Information Systems Research* 15, 1 (2004), 37–59.
9. Sunyaev, A. and Chorny, D. Supporting chronic disease care quality: Design and implementation of a health service and its integration with electronic health records. *ACM Journal of Data and Information Quality* 3, 2 (2012), 1–21.

Ali Sunyaev (sunyaev@wiso.uni-koeln.de) is an assistant professor in the Department of Management, Economics, and Social Sciences at the University of Cologne in Germany.

Stephan Schneider (schneider@wiso.uni-koeln.de) is a doctoral researcher in the Department of Information Systems and Systems Development at the University of Cologne.

Copyright held by author.

Viewpoint

The Explosive Growth of Postdocs in Computer Science

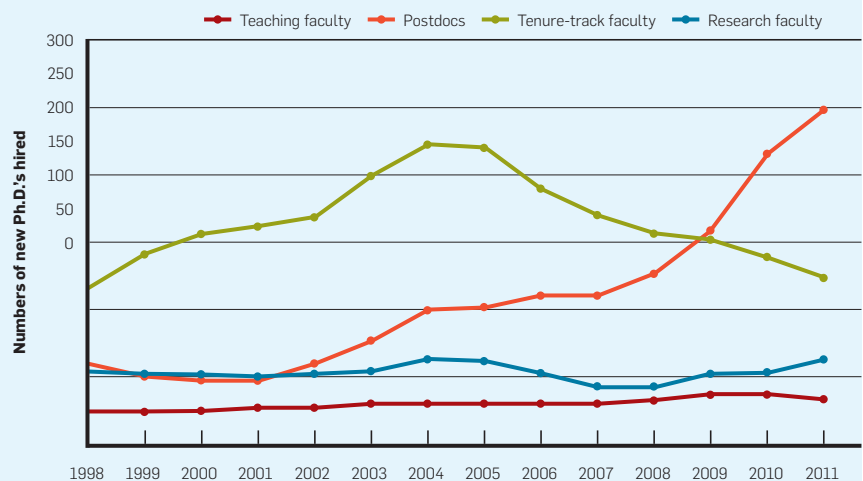
Considering the factors influencing the recent rapid increase in the number of postdoctoral positions in computer science.

THE NUMBER OF postdoctoral researchers in computer science slightly more than doubled in the decade ending in 2008. It is troubling that the number has doubled again in the subsequent three years. This changes the demographics of the academic computing research enterprise, in particular. In this Viewpoint, I discuss some of the different facets of this troubling trend.

The chart shown in the accompanying figure plots data from the 2012 Taulbee survey^a excluding data on graduates going into industry and those in the Taulbee “other” category. The data indicates tenure-track faculty positions for new Ph.D.’s have declined steadily since 2004 from 224 to 124 in 2011 while the number of postdoc positions greatly increased. In 2011, new Ph.D. graduates accepted twice as many postdoc positions as tenure-track positions. Around 2003 there were approximately 2.5 times more tenure-track positions than postdoc positions.

Most graduate departments primarily train Ph.D. students for a tenure-track

Hiring of computer science Ph.D.’s in academia, as a three-year rolling average 1998–2011.



faculty position in a research institution, though there is now some diversity in career path training in some universities. The Taulbee statistics document that most graduates will not achieve an academic tenure-track position. A mere 7% of graduates were hired directly into tenure-track faculty positions in 2011. Teaching faculty and research faculty have roughly stayed constant.

In past years computer science has been notable for the low level of postdocs. The National Science Board reports that in 2006, of the total number of postdocs in science and engineering

disciplines, approximately 1% were in computer science. Engineering, physics, and chemistry each had between 4% and 9.5% of the total.³

The Computing Community Consortium created the CIFellows Program to provide postdocs with funds from the National Science Foundation during the economic downturn at the end of the last decade in order to retain Ph.D.’s in research and teaching when universities dramatically curtailed hiring. With an improving economy that program has served its purpose and is concluding. The three-year program

^a Unless otherwise documented, the statistics quoted in this Viewpoint are extracted from the Taulbee survey, an annual survey of Ph.D.-granting departments in computer science, computer engineering, and information systems conducted by the Computing Research Association.



Association for
Computing Machinery

ACM Conference Proceedings Now Available via Print-on-Demand!

*Did you know that you can
now order many popular
ACM conference proceedings
via print-on-demand?*

Institutions, libraries and individuals can choose from more than 100 titles on a continually updated list through Amazon, Barnes & Noble, Baker & Taylor, Ingram and NACSCORP: CHI, KDD, Multimedia, SIGIR, SIGCOMM, SIGCSE, SIGMOD/PODS, and many more.

**For available titles and
ordering info, visit:
librarians.acm.org/pod**



funded 127 postdocs, a small number in relation to the postdoc growth in computer science.^b

Biomedical sciences have gotten very badly out of balance with respect to the number of postdocs and the long delay before they, on average, take permanent positions. The average age of first-time principal investigators obtaining R01-equivalent research funding (the major source of support for young investigators in the biomedical sciences) from the National Institutes of Health has risen to 44 years as of 2011, up from about 36 years in 1980.² To what degree will computer science move in the same direction?

The Postdoc Experience

A postdoc position is a *training* opportunity in which a person who has just completed a Ph.D. can deepen his or her expertise and research skills for a short period of time, en route to a permanent position. A postdoc—the person—may accept such a position for different reasons: to work under the tutelage of a specific expert, perhaps in a more highly regarded institution than his or her Ph.D.-granting university; to gain exposure in a related area; to remain research active while waiting for a faculty position; or to change fields altogether. The postdoc can have reasons that go beyond “training.” For example, the postdoc may wish to strengthen a research portfolio in anticipation of a competitive job search or to synchronize job search timing with that of a spouse.

A postdoc experience can genuinely advance an individual’s career: sharpen research skills, insight and knowledge; permit increase in publications; and enlarge the group of peers who know and respect the individual’s work.

The postdoc experience also has negatives. First, postdocs in academia are paid at a rate that is substantially lower than close peers. Taulbee data indicates the average (not starting) nine-month salary for an assistant faculty member was \$90,000 in 2010. A research faculty member was paid an average of \$81,000, and a teaching fac-

ulty member was paid an average of \$70,000. At the same time the comparable postdoc salary was \$50,000. In some universities postdocs have fewer or lesser benefits in areas such as health care, retirement, access to childcare, and access to wellness centers.

Second, the postdoc generally focuses only on their individual postdoc objectives without other distractions and responsibilities. Unless there is good mentoring and a strong collegial relation around a postdoc, that person could become isolated. In some cases, postdocs are directed to work on research projects or take on teaching obligations that do not advance their long-term career trajectories, simply because they are cheap labor for their advisors.

Third, the postdoc is mature in her or his intellectual power. Yet, at most universities and in industry, postdocs are isolated from participation in the discussions, much less the decisions, that set the future of the organization. Typically, postdocs cannot be principal investigators on grant proposals.

Last, the postdoc position is taken shortly after the degree. This is most often the time of life when couples start families. A postdoc position is not permanent; therefore the individual must do another job search, and typically will move from one geographic locale to another with the associated career disruption, personal disruption, and expense. Relocation is more difficult for women and men who are nurturing a young family. Delaying the bearing of children has health implications for women. In summary, there are distinct downsides to the postdoc experience.

Balance in the Academic Research Enterprise

The dramatic increase in postdocs changes the overall balance in the number of participants of different kinds in the academic research enterprise, that is the number of tenure-track faculty, graduate students, research faculty, teaching faculty, and postdocs. What effect does that have on other members of the enterprise?

When a recently graduated Ph.D. moves to a new research project, that person brings fresh ideas and even different assumptions about research. It is possible that a rapidly flowing pipeline of postdocs moving through a re-

^b The author has been a principal investigator on all the grants supporting the Computing Community Consortium and has been a member of its governing Council from founding to date.

search organization introduces new knowledge, broader interdisciplinary knowledge, more vitality, and a propensity to challenge assumptions—all in a way that students and the permanent faculty do not.

Perhaps the increase in postdocs is due to funding agencies insisting on project milestones that have a short duration or even a development, versus research, character. Perhaps postdocs are used to help manage research performance against tighter grant constraints.

One can make the simplistic assumption that the total computing research budget is fixed and does not increase or decrease with the number of postdocs. More postdocs means that fewer graduate students can be supported. Principal investigators make a choice among the categories—research faculty, students, and postdocs—and to a lesser degree to tenure-track faculty as they craft proposals.

Alternatively, the increase in postdocs may indicate a maturation of the field. Postdocs are more prevalent in math and the physical sciences. Postdoc positions are historically more prevalent in theoretical areas of computer science, as well.

Certainly, the explosive increase of postdocs in the life sciences is traceable back to funding. When the NIH budget doubled during the 1990s, either postdocs were the simplest way to rapidly expend the funding with no permanent obligations, or they were a way to deal with the overproduction of Ph.D.'s relative to tenure-track faculty positions. It is not clear that the life sciences *expected* to increase the number of postdocs so dramatically.

I have two principal concerns. First, I am concerned that our field is redefining “career progression” for researchers by expecting Ph.D. graduates to accept one to two years (or even more) of postdoc training before attaining a first, independent, permanent position. If that is necessary, then the community should clearly articulate why. I see no evidence that the increase derives from today's students not being as prepared as in the past.

Second, I am concerned that some academic departments and research laboratories are redefining hiring criteria so the quality of a candidate's

The increase in the number of postdocs is a major change for our field. We should manage it thoughtfully.

work is correlated with the number of publications consistent with a postdoc, rather than a number of publications consistent with a recent Ph.D. degree. Over the decades we have built a vibrant industrial and academic computer science research enterprise based on hiring for quality, not hiring for extensive publication.

I conclude that the increase in postdocs derives from desires of the research organizations to meet their own changed objectives, for example, in research project management and hiring or from a shortage of supply of high-quality Ph.D. students, not from a perception of training shortfalls among Ph.D. graduates.

Community Action

I recommend the community, with leadership from the Computing Research Association, take action.

- ▶ Get the data. Taulbee surveys do not provide all the information needed to understand the increase in postdocs and their later career paths.

- ▶ Understand this trend and articulate clearly what constitutes the best balance among different categories of positions in the research community going forward.

- ▶ Better manage the postdoc experience to deliver high value to the postdocs themselves.

Postdocs are frequently isolated, and sometimes benignly neglected. If the field requires many more postdocs for its own purposes, then the sponsoring departments and laboratories—not simply the mentors—should take responsibility to ensure the experience substantially contributes to the postdoc still in training. NSF now requires a postdoc training supplement to pro-

posals that fund postdocs. A postdoc should be given:

- ▶ A mentor who provides adequate guidance;


- ▶ A supportive set of colleagues in tenure-track faculty, other postdocs and research faculty who provide the postdoc with rich and frequent intellectual interaction;

- ▶ Skills in conducting research from proposal preparation to presentation and research group management; and

- ▶ Career development support with thoughtful exposure to alternative career paths, for example industrial research and development positions and bridges to other fields where computer science has a substantive role to play.

The sponsoring department or research laboratory—not only the mentor—should proactively support and enforce high standards for all four.

In addition, the Ph.D. advisor—who typically regards a Ph.D. student as an “intellectual descendent”—may want to continue vigilance on behalf of the Ph.D. student who accepts a postdoc position. The advisor should monitor to ensure the host organization for the postdoc is providing an experience that will genuinely advance the postdoc's career.

The increase in the number of postdocs is a major change for our field. We should manage it thoughtfully. A document outlining “best practices” for nurturing postdocs in computer science can be found on the Computing Research Association website (http://cra.org/resources/bp-view/best_practices_memo_computer_science_postdocs_best_practices/). 

References

1. Computing Research Association, Taulbee Survey, 1996–2011; <http://www.cra.org/resources/taulbee/>.
2. National Institutes of Health Office of Extramural Research. NIH Extramural Data Book: Average Age of Principal Investigators; http://report.nih.gov/NIH_Investment/PDF_sectionwise/NIH_Extramural_DataBook_PDF/NEDEB_SPECIAL_TOPIC-AVERAGE_AGE.pdf.
3. National Science Board, Science and Engineering Indicators 2010; <http://www.nsf.gov/statistics/seind10/>.

Anita Jones (jones@virginia.edu) is University Professor Emerita and past Department Chair and faculty member in the Department of Computer Science at the University of Virginia in Charlottesville.

I thank Erwin Gianchandani for his thoughtful discussion of this issue as we drafted the first Computing Research Association paper on the subject of postdocs (<http://cra.org/postdocs/>) and for the preparation of the data used in the figure in this Viewpoint.

Copyright held by author.

Article development led by **acmqueue**
queue.acm.org

Our authentication system is lacking. Is improvement possible?

BY WILLIAM CHESWICK

Rethinking Passwords

THERE IS AN authentication plague upon the land. We have to claim and assert our identity repeatedly to a host of authentication trolls, each jealously guarding an Internet service of some sort. Each troll has specific rules for passwords, and the rules vary widely and incomprehensibly.

Password length requirements vary: Dartmouth wants exactly eight characters; my broker, six to eight; Wells Fargo, eight or more. Special characters are often encouraged or required, but some characters are too special: many disallow spaces, single or double quotes, underlines, or hyphens. Some systems disallow certain characters at the beginning of the password; dictionary checks abound, including foreign language dictionaries.

Sure, brokerage, bank, and medical sites need to protect accounts from unauthorized use. So do shopping sites such as Amazon. An email account might be just as important: ask Sarah Palin. The value of an account can change over time: perhaps a new online store is added to a previously unimportant account.

Authentication may be more important to the service provider than to the client: do I care if someone gains access to my newspaper account? (The terms of use undoubtedly say I am supposed to care, but I do not.) In this case, the newspaper's very requirement of a password is a nuisance, and the password-"strengthening" rules just increase my annoyance. The marketplace does work here: studies show that competitive pressure tends to force sites toward simpler passwords.⁴

Not only do these authentication rules vary widely, the rules themselves are often considered to be part of the security secret and not available at login time, when a hint about the rules would be helpful. I call these *eye-of-newt* password rules: they remind me of the formulae for magic potions from Shakespeare. They are often particular, exacting, and sometimes difficult to satisfy. Can you think of a long passphrase that does not repeat any character more than four times?

The problem is emergent: if we had only one account, authentication would be much easier. But an active Internet user can have one- or two dozen accounts, some important, some not. These authentication trolls bother most online users, and it is easy to elicit a litany of complaints from casual users.

Many of today's rules are rooted in the deep past of security concerns, when access, threats, and targets were different. Many of these ideas were presented in the *Password Management Guideline*, (Technical Report CSC-STD-002-85), published by the Department of Defense Computer Security Center (DoD CSC) in 1985.² Known as the Green Book, this report was one of the Rainbow Series of books put out by the U.S. government in the 1980s and 1990s. Its advice was good at the time, and much of it still holds up, but many of our password aphorisms come from dated assumptions about threats and technology.

This is not a criticism of the original authors or their document: no sensible

security person would expect these rules to stand unamended for decades. The lore has simply not kept up with the threats and vulnerabilities.

The Green Book In Today's World

The *Password Management Guideline* came out shortly after the more-famous Orange Book (Trusted Computer System Evaluation Criteria). The Green Book was the DoD's management guideline for access to classified or sensitive government computers. It is also the basis for most of the current password rules. Most computer access at the time was either via local batch processing (with cards!) or through local or remote serial lines using terminals. The PC and Macintosh were available, but they were not especially relevant to secure computing and certainly were not networked.

Here is an important note found early in the report:

Because it is anticipated that diverse user communities will adopt this guideline, all recommendations are presented in general rather than specific terminology...Where features require the setting of a specific value (for example, password maximum lifetime), it is suggested that these be designed as parametric settings leaving the determination of exact values to local security management...

The question for today's security specialists is, what still makes sense from the 1985 guidelines? The current authentication mess suggests that we have not kept up with this task. Perhaps this article will spur some rethinking along these lines.

The DoD report offered specific advice about authentication and passwords. It stated that in a password-based authentication mechanism implemented on an ADP (automated data processing) system, passwords are vulnerable to compromise because of five essential aspects of the password system:

1. *A password must be initially assigned to a user when enrolled on the ADP system.* This rule is still fine. Many sites have used a standard password for

the initial password but skipped the requirement to force a change to the default password—an attacker could simply try a number of accounts with the default password to break into a system. The same held for some reset password schemes. One solution that encouraged a change of default password was to set the default or recovery password to “I am stupid.”

2. *A user's password must be changed periodically.* I will discuss this in more detail later.

3. *The ADP system must maintain a password database.* This rule is still fine.

4. *Users must remember their passwords.* It turns out this rule is unreasonable, especially for machine-gen-

erated passwords. These passwords are simply not that memorable, and to memorize multiple ones for a long time is beyond the abilities of most people. Also, people logged into many fewer systems in 1985.

5. *Users must enter their passwords into the ADP system at authentication time.* Rule 5 is incomplete: it is only single-factor authentication. The alternatives were undoubtedly well known to the authors, but probably too expensive for general deployment. I suspect that a remark to this effect at that time might have changed our world.

Furthermore, according to the report:
► *Users should be able to change their own passwords.* This is a good idea.



Some systems in the deep past did not allow this.

► *Passwords should be machine-generated rather than user-created.* It is true that machine-generated passwords tend to be much stronger: the work factor needed to crack them is easy to compute and noncontroversial. Not so for human-created passwords, where a sea of associations and language rules greatly reduces the search space.

► *Certain audit reports (for example, date and time of last login) should be provided by the system directly to the user.* This gives the user an opportunity to spot unauthorized accesses. The practice was widely adopted in Unix systems with the `login(1)` command. It is still a fine idea.

► *User ID is a unique symbol or character string that is used by an ADP system to uniquely identify a user.* The security provided by a password system should not rely on secrecy of the user's ID. This is a typical cryptographic assumption, that only the key is secret, not the user ID. (I wish this were true for the Social Security Number in the U.S.) Obscuring the user ID can be a useful barrier to wholesale attacks, however, especially against massive online systems.

► *Throughout the lifetime of an ADP*

system, each user ID should be assigned to only one person. In other words, do not share accounts and their associated passwords. This is still a good idea for important accounts, because it may aid in logging and attribution. This can be especially important for shared accounts when a marriage is failing: former partners can be very nasty.

► *All user IDs should be revalidated periodically.* This is a good idea, but it is rarely implemented. Many break-ins have occurred on unused or fallow accounts. Some systems implement a “we-haven't-seen-you-in-a-while” increase in authentication requirements, a good idea. A modern version includes stronger authentication when connections come from unusual locations or IP addresses.

► *The simplest way to recover from the compromise of a password is to change it.* Ah, the good old days! This is just wrong now. Once an account is compromised, the rot sets in and spreads through further attacks and transitive trust. Other accounts are attacked with the same password, often successfully. Bank accounts are drained (at least temporarily—personal exposure has declined on this,³ plasma screens ordered, billing addresses changed, and identities stolen.)

On Unix/Linux personal accounts, a stolen password is just the beginning. Systems are rooted, backdoors installed, and, often, other security weaknesses are fixed. SSH (secure shell) clients are installed to capture other passwords. It tends to be easier to root a Unix host given a user account. Table 1 is a sample of the number of SUID(root) programs on a few

sample operating systems, found with the command `find / -user root -perm -4000 -print`.

Each of the examples in Table 1 is a potential root compromise, and an attacker can often find at least one.

► *Passwords should be changed* on a periodic basis to counter the possibility of undetected password compromise.

The most obvious threat to the security provided by a password system is from the compromise of passwords. The greater the length of time during which a password is used for authentication purposes, the more opportunities there are for exposing it. In a useful password system, the probability of compromise of a password increases during its lifetime.

This section refers to Green book Appendix C, which gets to the meat of password strength and lifetime in the face of dictionary attacks. Several simple formulae are offered (an ASCII layout and typos makes the math more difficult to follow in the online versions), and results computed for a typical case of the time.

The goal is to resist a year's worth of dictionary attacks with a cracking probability of 10^{-6} (or 10^{-20} for sensitive systems). To give one of the report's examples, a nine-character password of only uppercase letters can resist a yearlong dictionary attack over a 30-character-per-second terminal session, assuming 8.5 guesses per minute. The report offers similar computations for uppercase alphanumeric characters and words selected from a 23,300-entry dictionary of English words from four to six characters in length. The authors admit a much higher guessing rate if a file on hand is protected by a password.

Let's plug in the numbers for a modern dictionary attack using 100 million and seven billion trials per second. The first is an easy rate for a multicore machine running on typical password-hashing algorithms. The second rate is claimed for attacks implemented on modern GPUs by a commercial source. These are somewhat conservative numbers in an age of multicore processors, clusters of computers, and botnets. If you think they are too aggressive, wait a year. Table 2 shows the cracking time and password change rates for some variations.

Table 1. Number of `setuid(root)` programs found with `find`.

System	Number of programs
Linux (Ubuntu)	19
FreeBSD 9.0	38
OSX 10.8.2	34
FreeBSD 7.2	46

Table 2. Cracking time and password change rates.

Scheme	Search space (bits)	7 billion trials/second		10 million trials/second	
		Cracked in	Change time	Cracked in	Change time
8-character, full alphanumeric	47.6	0.36 mins.	31.19 ms.	252.71 days	21.83 sec.
8-character, eye-of-newt	52.3	9.25 days	799.40 ms.	17.74 years	559.58 sec.
11-character, eye-of-newt	71.9	20,390 years	7 days	1.43E+07 years	14.3 years
13-character, full alphanumeric	77.4	906,123 years	331 days	2.32E+11 years	634 years
12 character, eye-of-newt	78.5	1,896,229 years	692 days	4.84E+11 years	1,327 years

The second scheme shown in the table is tougher than passwords considered secure these days: it is eight *random* characters chosen from the 93 characters found on a keyboard (a bit more than eye-of-newt). This strong password needs to be changed every *31 milliseconds* for security purposes. (My crude spreadsheet for exploring this is available on my Web site.¹)

The last two schemes in the table roughly meet the criteria of this document: the password may be changed annually without risking more than a one-in-a-million chance of compromise after a yearlong dictionary attack. These correspond to a work factor of 77–79 bits, which might surprise you as being much larger than typical password strengths actually required, usually from 20 to the mid-40s.³ The added bits come from the requirement of 10^{-6} guessing success probability, which adds 20 bits to the password length. (The spec actually calls for a probability of 10^{-20} for classified access: that adds 66 bits!)


The one-in-a-million requirement is probably unreasonable. With an installation of very expensive brute-force hardware, I am unlikely to deploy it for a year to gain access to a high-value target if my chances of success are, say, 1%. On the other hand, history is full of examples of defenders underestimating the amount of work an attacker is willing to undertake.

Other Aphorisms


► *Do not use the same password on multiple services.* This is still a very good idea, though I realize that it is a pain in the neck. If I break into your Facebook account, then I am going to try that same password on LinkedIn, Gmail, iTunes, and so on. This attack works beautifully, because most people do not follow this rule.

Most practitioners who do follow this rule use a basic password, modified by some service-dependent portion. If that variable portion is obvious, they probably should not bother. In this case, it would probably be better to choose different, strong passwords and ignore the next piece of advice.

► *Do not write your passwords down.* This rule depends very strongly on your threat model: what are you afraid of? In the deep past, many attacks came from



Once an account is compromised, the rot sets in and spreads through further attacks and transitive trust. Other accounts are attacked with the same password, often successfully.



fellow students, co-workers, family members, and on-site spies. The movie trope of checking for Post-It notes around the desk worked, and still does.

Writing your passwords down, however, is probably much safer than using the same password on multiple machines. In most cases today, the attacker does not have to be present to win. Your machine can be compromised from very far away. Or the attacker leaves infected USB thumb-sticks in the company parking lot. The check-the-Post-It attack is much less common than networked hacking attacks.

Of course, there is no need to make it too easy. Write down a comment or variation on the password that is sufficient to remind you of the real password. Sometimes I find a reminder of the particular site's eye-of-newt rule is enough.

Password wallets are a terrific idea for storing passwords, but they get you back in the game of storing secrets on possibly unsecured computers with network access. The yellow pad in your office is probably more secure.

► *Change your passwords often.* This is often enforced by the authentication service, and it is generally a bad idea—and not useful. A good, strong password that you can remember is difficult to create and probably difficult to remember, especially if there are different passwords for different accounts. When a password is changed by force, all that goodness goes away, requiring a whole new effort.

This can be a particular problem for rarely used passwords. For example, corporate-provided health care in the U.S. requires employees to review and make changes to coverage options annually. These systems require strong authentication and tend to be used exactly once a year, so to remember the password at all, I either write it down or rely on the password-recovery scheme. On some systems, I have cycled through several strong passwords over a longer period than the authentication server remembers. Those really good passwords are too good to let go.

What We Have Learned

It is simply poor engineering to expect people to choose and remember passwords that are resistant to dictionary attacks. User training does not work:

people will write down their passwords regardless.

Fortunately, dictionary attacks are rarely the problem. They are completely frustrated by getting out of the game: limit the number of attempts to a handful, then disable the account. Multiple-factor authentication and better recovery from compromise have also helped out.

This is not a new idea. I got my first bank ATM card in the early 1970s; it had a four-digit PIN. I do not recall if I was allowed to select the PIN myself, but it did not matter: it was my only PIN, and the service was unique and useful enough that I committed the PIN to memory. If I forgot it, the card would be eaten, or the account locked. This policy is still used in the U.S. banking system some four decades later, proof that it is working. It is also not a rare solution. Most authentication systems lock a user out after several tries.

More importantly, the threats have changed. Dictionary attacks on passwords are not nearly the problem they used to be. Today's threats include:

- ▶ Keystroke loggers record any password, no matter how complex.

- ▶ Phishing sites capture the passwords of the unwary, and it is very easy to be unwary. The mail reader should present any URL found in an email with red flags and warnings, especially if it refers to an unfamiliar domain.

- ▶ Password files from poorly protected servers spill our secrets across the Internet, eye-of-newt passwords included.

- ▶ Sites that have passed state-of-the-art security audits are later found to have been leaking credit card information for years. Best-in-practice may be good enough for the lawyers, but it really is not solving very hard security problems.

Client systems are hardly secure—we have built our houses on sand. Why should any mouse click present a security threat?

Dictionary attacks can be launched on password wallets, SSH agent passphrases, PGP (Pretty Good Privacy) key rings, and stolen password databases. For strong passwords, words, rather than eye-of-newt strings, are easier to type and remember. From the Brown corpus's top 23,300 common English words, I generated several random passphrases in the spirit of STD-002 and xkcd:⁵

- ▶ fooled otherwise faustus
- ▶ exclaimed democrat cruz
- ▶ deauville attaches ornamented
- ▶ acutely jeep pasha

These give a search space of more than 43 bits, matching the estimated strength of today's strongest passwords. They also offer a chance to expand one's vocabulary. Alas, they probably fail most eye-of-newt rules.

Suggestions

My dream is that authentication might become a lot less odious, maybe even fun. Passwords and passphrases should be easier to type and include automatic correction for typing and “tapographical” errors (on smart phones). This can be done without loss of security.

Why do the eye-of-newt rules remain? Account unlocking is a problem, requiring relatively costly or unsecured secondary authentication efforts. In some cases, it would be appropriate to have someone else—for example, an authorized spouse on a shared back account—enable the temporary authentication and subsequent password change. “Honey, I did it again” could be much easier than getting through to an 800 number on a weekend.

It would be nice to have more than one way to log into a site, each way having about the same strength. This gives the users a choice of authentication methods, with other methods as a backup login. (Mother's maiden name is *not* what I am talking about here. Secondary passwords tend to be much weaker and should not be used. Security history is full of attacks that force the defender to drop back to secondary, less effective defenses.)

If one tries the same password twice in a session, that should not count as two tries. We all make, or suspect that we make, typographical errors. Did I enter that password correctly? I will try again more carefully. This should not count as a second wish for the password troll.

Conclusion

I am not optimistic that these changes will happen rapidly, or even at all. There is a huge installed base out there. “We do the same thing as everybody else” is an effective legal defense against malfeasance, so why change things? (I hate the word *legacy!*)

Authentication systems are vital, and changes to them can produce widespread and embarrassing failures. It is not clear that easier authentication would provide a market advantage. Is a company less secure than another company because it is easier to log into? Will it gain market share by doing so?

In spite of all this, the system seems to be working. We are leaking military and industrial secrets to attackers all over the world, but millions of people use the Internet successfully every day, and it is an important part of the world's economy. Somehow, we get by.

Finally, I would like to see these systems engineered such that the user needs to remember only one security maxim: *Don't be a moron*. Do not pick a password that someone who knows you can guess in a few tries, or that someone watching you type can figure out easily.

Unlike the eye-of-newt password rules, this last rule makes sense to the casual user and is easy to remember. All we have to do is engineer the rest to be reasonably secure. □

Related articles on queue.acm.org

Security - Problem Solved?

John Viega

<http://queue.acm.org/detail.cfm?id=1071728>

Building Secure Web Applications

George V. Neville-Neil

<http://queue.acm.org/detail.cfm?id=1281889>

LinkedIn Password Leak: Salt Their Hide

Poul-Henning Kamp

<http://queue.acm.org/detail.cfm?id=2254400>

References

1. Cheswick, W. 2012; <http://www.cheswick.com/ches/papers/std-002-results.xls>; and <http://www.cheswick.com/ches/papers/std-002-results.numbers>.
2. Department of Defense Computer Security Center. *Password Management Guideline*, 1985. Technical Report CSC-STD-002-85.
3. Florêncio, D. and Herley, C. Is everything we know about password stealing wrong? *IEEE Security and Privacy* 99 (2012). DOI 10.1109/MSP.2012.57.
4. Florêncio, D. and Herley, C. Where do security policies come from?. In *Proceedings of the Sixth Symposium on Usable Privacy and Security* (2012). ACM, NY, DOI 10.1145/1837110.1837124. <http://doi.acm.org/10.1145/1837110.1837124>.
5. xkcd; <http://xkcd.com/936/>.

William “Ches” Cheswick was formerly with Bell Labs, Lumeta Corporation, and AT&T Shannon Lab. One of his current projects is promoting better passwords. The earliest password he can remember using is “polpis,” a location in Nantucket. He now uses multiple random words for important accounts, and writes stuff down.

© 2013 ACM 0001-0782/13/02

The USE method addresses shortcomings in other commonly used methodologies.

BY BRENDAN GREGG

Thinking Methodically about Performance

PERFORMANCE ISSUES CAN be complex and mysterious, providing little or no clue to their origin. In the absence of a starting point—or a methodology to provide one—performance issues are often analyzed randomly: guessing where the problem may be and then changing things until it goes away. While this can deliver results—

if you guess correctly—it can also be time consuming, disruptive, and may ultimately overlook certain issues. This article describes system-performance issues and the methodologies in use today for analyzing them, and it proposes a new methodology for approaching and solving a class of issues.

Systems-performance analysis is complex because of the number of components and their interactions in a typical system. An environment may be composed of databases, Web servers, load balancers, and custom applications, all running upon operating systems—either bare-metal or virtual. And that is just the software. Hardware and firmware, including external storage systems and

network infrastructure, add many more components to the environment, any of which is a potential source of issues. Each of these components may require its own field of expertise, and a company may not have staff knowledgeable in all the components in its environment.

Performance issues may also arise from complex interactions between components that work well in isolation. Solving this type of problem may require multiple domains of expertise to work together.

As an example of such complexity within an environment, consider a mysterious performance issue we encountered at Joyent for a cloud-computing customer: the problem ap-

peared to be a memory leak, but from an unknown location. This was not reproducible in a lab environment with the components in isolation. The production environment included the operating system and system libraries, the customer's own application code written in node.js, and a Riak database running on the Erlang VM (virtual machine). Finding the root cause required knowledge of the customer's code, node.js, Riak, Erlang, and the operating system, each of which was provided by one or more different engineers. The problem turned out to be in the system library, identified by engineers with operating-systems expertise.

Another complicating factor is that “good” or “bad” performance can be subjective: what may be unacceptable latency for one user may be acceptable for another. Without a means of clearly identifying issues, it can be difficult to know not only if an issue is present, but also when it is fixed. The ability to measure performance issues—for example, as an expression of response time—allows them to be quantified and different issues ranked in order of importance.

Performance-analysis methodology can provide an efficient means of analyzing a system or component and identifying the root cause(s) of problems, without requiring deep expertise. Methodology can also provide ways of identifying and quantifying issues, allowing them to be known and ranked.

Performance texts have provided methodologies for various activities, such as capacity planning,^{1,16} benchmarking,¹⁸ and modeling systems.^{7,8,10} Methodologies for finding the root causes of performance issues, however, are uncommon. One example is the drill-down analysis method introduced in Solaris Performance and Tools,¹³ which describes a three-stage procedure for moving from a high-level symptom down to analyzing the cause. These texts have typically covered analysis by use of ad hoc checklists of recent tips and tuning, and by teaching operating-systems internals and tools.^{2,11,12,15} The latter allows performance analysts to develop their own methodologies, although this can take considerable time to accomplish.

Ad hoc performance checklists have been a popular resource. For example, *Sun Performance and Tuning*² includes

“Quick Reference for Common Tuning Tips,” which lists 11 tips, intended to find disk bottlenecks, network file system (NFS), memory, and CPU issues, and is both easy to follow and prescriptive. Support staff groups often use these lists because they provide a consistent check of all items, including the most egregious issues. This approach poses some problems, however. Observability is limited to the specific items in the list, and they are usually point-in-time recommendations that go out of date and require updates. These checklists also focus on issues for which there are known fixes that can be easily documented, such as the setting of tunable parameters, but not custom fixes to the source code or environment.

In this article I summarize several other methodologies for systems-performance analysis, including the USE method, which is explained in detail. I begin by describing two commonly used *anti*-methodologies—the blame-someone-else anti-method and the streetlight anti-method—that serve as comparisons with later methodologies.

The “Blame-Someone-Else” Anti-Method

The first anti-methodology follows these simple steps:

1. Find a system or environment component you are not responsible for.
2. Hypothesize that the issue is with that component.
3. Redirect the issue to the responsible team.
4. When proven wrong, go back to step 1.

For example, “Maybe it's the network. Can you check with the network team to see if they have had dropped packets or something?”

Instead of investigating performance issues, this methodology makes them someone else's problem, which can be wasteful of other teams' resources. A lack of data analysis—or even data to begin with—leads to the hypothesis. Ask for screen shots showing which tools were run and how their output was interpreted. These can be taken to someone else for a second opinion.

The Streetlight Anti-Method

While running tools and collecting data is better than wild hypotheses, it is not sufficient for effective performance

analysis, as shown by the streetlight anti-method. This is the absence of any deliberate methodology. The user analyzes performance by selecting observability tools that are familiar, found on the Internet, or found at random and then seeing whether anything obvious shows up. This hit-or-miss approach can overlook many types of issues.

Finding the right tool can take a while. The most familiar tools are run first, even if they do not make the most sense. This is related to an observational bias called the *streetlight effect*,¹⁷ named after a parable:

A policeman sees a drunk hunting for something under a streetlight and asks what he is looking for. The drunk says he has lost his keys. The policeman cannot find them either, and asks if he lost them under the streetlight. The drunk replies: “No, but this is where the light is best.”

The performance equivalent would be looking at top(1), not because it makes sense but because the user does not know how to read other tools.

Learning more tools helps but is still a limited approach. Certain system components or resources may be overlooked because of a lack of observability tools or metrics. Furthermore, the user, unaware that the view is incomplete, has no way of identifying “unknown unknowns.”

Better performance-analysis methodologies are available that may solve issues before you run any tools at all. These include the problem statement method, workload characterization, and drill-down analysis.

Problem Statement Method

The problem statement method, commonly used by support staff for collecting information about a problem, has been adapted for performance analysis.⁹ It can be the first methodology attempted for performance issues.

The intent is to collect a detailed description of the issue—the problem statement—that directs deeper analysis. The description on its own may even solve the issue. This is typically entered into a ticketing system by asking the following questions:

- ▶ What makes you think there is a performance problem?
- ▶ Has this system ever performed well?

- ▶ What has changed recently? (Software? Hardware? Load?)
- ▶ Can the performance degradation be expressed in terms of latency or runtime?
 - ▶ Does the problem affect other people or applications (or is it just you)?
 - ▶ What is the environment? What software and hardware is used? Versions? Configuration?

These questions may be customized to the environment. While the questions may seem obvious, the answers often resolve a class of issues, requiring no deeper methodologies. When that is not the case, other methodologies can be called into service, including workload characterization and drill-down analysis.

The Workload Characterization Method

The workload can be characterized by answering questions such as:

- ▶ Who is causing the load? Process ID, user ID, remote IP address?
- ▶ Why is the load being called? Code path?
- ▶ What are other characteristics of the load? IOPS, throughput, type?
- ▶ How is the load changing over time?


This helps to separate *problems of load from problems of architecture*, by identifying the former.

The best performance wins often arise from eliminating unnecessary work. Sometimes these bottlenecks are caused by applications malfunctioning (for example, a thread stuck in a loop) or bad configurations (systemwide backups running during the day). With maintenance or reconfiguration, such unnecessary work can be eliminated. Characterizing the load can identify this class of issue.


The Drill-Down Analysis Method

Drill-down analysis involves peeling away layers of software and hardware to find the core of the issue—moving from a high-level view to deeper details. These deeper details may include examining kernel internals—for example, by using profiling to sample kernel stack traces, or dynamic tracing to examine the execution of kernel functions.

Solaris Performance and Tools¹³ provides a drill-down analysis methodology for system performance. It



“Good” or “bad” performance can be subjective: what may be unacceptable latency for one user may be acceptable for another. Without a means of clearly identifying issues, it can be difficult to know not only if an issue is present, but also when it is fixed.



follows three stages:

- ▶ *Monitoring.* This continually records high-level statistics over time across many systems, identifying or alerting if a problem is present.
- ▶ *Identification.* Given a system with a suspected problem, this narrows the investigation to particular resources or areas of interest using system tools and identifying possible bottlenecks.
- ▶ *Analysis.* This stage provides further examination of particular system areas, identifying the root cause(s) and quantifying the issue.

The analysis stage may follow its own drill-down approach, beginning with applications at the top of the software stack and drilling down into system libraries, system calls, kernel internals, device drivers, and hardware.

While drill-down analysis often pinpoints the root cause of issues, it can be time consuming, and when drilling in the wrong direction, it can waste a great deal of time.

The Need for a New Methodology

I recently analyzed a database performance issue on the Joyent public cloud, which began with a ticket containing a problem statement as described in the previous section. The statement indicated that there was a real issue that needed deeper analysis.

The issue had been intermittent, with some database queries taking seconds to complete. The customer blamed the network, hypothesizing that the query latency was caused by dropped network packets. This was not a wild hypothesis, as the ticket included output from `ping(1)` showing occasional high latency; `ping(1)` is a common and familiar tool, however, and with no other supporting evidence, this seemed to be an example of the streetlight anti-method.

The support team ran tools to investigate the network in much more detail, including examining TCP/IP stack network counters, without finding any problems. This analysis took time because there are dozens of such statistics, some of which are difficult to interpret and must be examined over time to look for correlations. While logged into the systems, the team also checked CPU usage vs. the cloud-imposed limits, following their own ad hoc checklist of common is-

sues. Their conclusion was that there was no issue while they were watching: the network and CPUs were fine.

At this point, many system components and tens of thousands of system statistics had not yet been checked, as they were assumed to be unrelated to the issue. Without a direction to follow, checking everything across all systems in the customer's cloud environment could take days. The analysis to date had not found any evidence of a real issue, which was discouraging.

The next step was to try dynamic tracing of the originally reported problem (network packet drops), in the hope of finding something that the standard network counters had missed. I have used the DTrace tool many times to perform drill-down analysis of the TCP/IP stack. This can provide many details beyond the standard network observability toolset, including inspection of kernel-dropped packets and internal TCP state. It still can take hours to catch intermittent issues, however. I was tempted to begin drill-down analysis from the database query latency, in case the issue was not network related, or to begin characterizing the database workload over time, in case the problem was caused by a burst of load, but these approaches are also time consuming.

Before beginning deeper analysis, I wanted to perform a quick check of all system components, not just the network and CPUs, to look for bottlenecks or errors. For this to be quick, it would need to check only a limited number of statistics per system, not the tens of

thousands available. And for this to be complete, it would need to check all components, including those that might be missed because they have no observability tools or statistics by default.

The utilization, saturation, and errors (USE) method provided one way of doing this. It quickly revealed that the database system was out of memory and was paging, and that the disks were occasionally running at saturation. Focusing troubleshooting efforts on networking early on had meant these areas were overlooked in the team's analysis. The real issues were in the system memory and disks, which were much quicker to read and interpret.

I developed the USE method while teaching classes in operating-systems performance. The goal was to help my students find common issues and to ensure that they were not overlooking important areas. I have used it successfully many times in enterprise and cloud-computing environments, but it does not solve all types of problems and should be treated as just one methodology in the toolbox.

The USE Method

The USE method is intended to be used early in a performance investigation, after the problem-statement method, to identify systemic bottlenecks quickly. It can be summarized as: *For every resource, check utilization, saturation, and errors.*

Resource in this case means all physical server functional components (CPUs, disks, buses, and so on) examined indi-

vidually. Some software resources can be examined using the same methodology, provided the metrics make sense.

Utilization is the percentage of time that the resource is busy servicing work during a specific time interval. While busy, the resource may still be able to accept more work; the degree to which it cannot do so is identified by *saturation*. That extra work is often waiting in a queue.

For some resource types, including main memory, utilization is the *capacity* of the resource that is used. This is different from the time-based definition. Once a capacity resource reaches 100% utilization, no more work can be accepted, and it either queues the work (saturation) or returns errors, either of which is identified by the USE method.

Errors in terms of the USE method refer to the count of error events. Errors should be investigated because they can degrade performance, and they may not be immediately noticed when the failure mode is recoverable. This includes operations that fail and are retried, as well as devices that fail in a pool of redundant devices.

In contrast to the streetlight anti-method, the USE method iterates over system resources instead of starting with tools. This creates a complete list of questions to ask, and only then searches for the tools to answer them. Even when tools cannot be found to answer the questions, the knowledge that these questions are unanswered can be extremely useful for the performance analyst: they are now “known unknowns.”

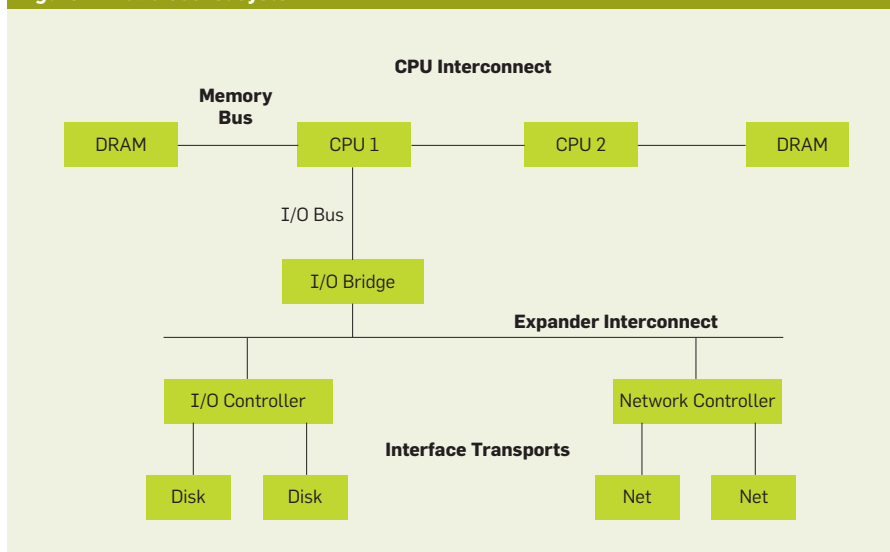
The USE method also directs analysis to a limited number of key metrics, so that all system resources are checked as quickly as possible. After this, if no issues have been found, you can turn to other methodologies.

The key metrics of the USE method are usually expressed as follows:

- ▶ Utilization as a percentage over a time interval (for example, one CPU is running at 90% utilization).
- ▶ Saturation as a wait queue length (for example, the CPUs have an average run queue length of four).
- ▶ Errors as the number of errors reported (for example, the network interface has had 50 late collisions).

It is also important to express the time interval for the measurement. Though it may seem counterintuitive, a

Figure 1. A two-socket system.



short burst of high utilization can cause saturation and performance issues, even though the overall utilization is low over a long interval. Some monitoring tools report utilization as five-minute averages. CPU utilization, for example, can vary dramatically from second to second, and a five-minute average can disguise short periods of 100% utilization and, therefore, saturation.

The first step in the USE method is to create a list of resources. Try to be as complete as possible. Here is a generic list of server hardware resources, with specific examples:

- ▶ **CPUs**—Sockets, cores, hardware threads (virtual CPUs).
- ▶ **Main memory**—DRAM.
- ▶ **Network interfaces**—Ethernet ports.
- ▶ **Storage devices**—Disks.
- ▶ **Controllers**—Storage, network.
- ▶ **Interconnects**—CPU, memory, I/O.

Each component typically acts as a single resource type. For example, main memory is a *capacity* resource, and a network interface is an I/O resource, which can mean either IOPS (I/O operations per second) or throughput. Some components can behave as multiple resource types—for example, a storage device is both an I/O resource and a capacity resource. Consider all types that can lead to performance bottlenecks. Also note that I/O resources can be further studied as *queueing systems*, which queue and then service these requests.

Some physical components can be left off your checklist, such as hardware caches (for example, MMU TLB/TSB, CPU Level-1/2/3). The USE method is most effective for resources that suffer performance degradation under high utilization or saturation, leading to bottlenecks; caches *improve* performance under high utilization.

Cache hit rates and other performance attributes can be checked after the USE method has been applied—that is, after systemic bottlenecks have been ruled out. If you are unsure whether to include a resource, go ahead and include it and then see how well the metrics work in practice.

Function Block Diagram

Another way to iterate over resources is to find or draw a function block diagram³ for the system. This type of diagram also

shows relationships, which can be very useful when looking for bottlenecks in the flow of data. Figure 1 is a generic diagram showing a two-socket system.

While determining utilization for the various buses, annotate each one on the functional diagram with its maximum bandwidth. The resulting diagram may pinpoint systemic bottlenecks before a single measurement has been taken. (This is also a useful exercise during hardware product design, while you still have time to change physical components.)

CPU, memory, and I/O interconnects are often overlooked. Fortunately, they are not commonly the cause of system bottlenecks. Unfortunately, when they are, the problem can be difficult to solve (maybe you can upgrade the main board or reduce load (for example, “zero copy” projects lighten memory bus load). At least the USE method takes interconnect performance into consideration. (See Analyzing the HyperTransport⁴ for an example of an interconnect issue identified in this way.)

Once you have your list of resources, consider the types of metrics you need for each (utilization, saturation, and errors). Table 1 lists some example resources and metric types, along with possible metrics (from generic Unix/Linux). These metrics can be expressed either as averages per interval or as counts.

Repeat for all combinations and include instructions for fetching each metric. Take note of metrics that are not currently available: these are the “known unknowns.” You will end up with a list of about 30 metrics, some of which are difficult to measure and some of which cannot be measured at all. Example checklists have been built for Linux- and Solaris-based systems.^{5,6}

Fortunately, the most common issues are usually found with the easier metrics (for example, CPU saturation, memory capacity saturation, network interface utilization, disk utilization), so these can be checked first.

Table 2 lists some examples of harder combinations. Some of these metrics may not be available from

Table 1. Resources and metric types.

Resource	Type	Metric
CPU	utilization	CPU utilization (ideally per CPU)
CPU	saturation	dispatcher queue length (aka run-queue length)
Memory	utilization	available free memory (systemwide)
Memory	saturation	anonymous paging or thread swapping (“page scanning” is another indicator)
Network interface	utilization	RX/TX throughput / max bandwidth
Storage device I/O	utilization	device busy percent
Storage device I/O	saturation	wait queue length
Storage device I/O	errors	device errors (“soft,” “hard”)

Table 2. Harder combinations.

Resource	Type	Metric
CPU	errors	correctable CPU cache ECC events or faulted CPUs (if the OS+HW supports that)
Memory	errors	failed malloc(s) (although this is usually because of virtual memory exhaustion, not physical)
Network	saturation	saturation-related NIC or operating-system errors (for example, Solaris “nocanputs”)
Storage Controller	utilization	depends on the controller; it may have a max IOPS or throughput that can be checked vs. current activity
CPU interconnect	utilization	per port throughput/max bandwidth (CPU performance counters)
Memory interconnect	saturation	memory stall cycles, high CPI (CPU performance counters)
I/O interconnect	utilization	bus throughput/max bandwidth (performance counters may exist on your hardware; for example, Intel “uncore” events)

standard operating-system tools. I often have to write my own software for such metrics, using either static or dynamic tracing (DTrace) or the CPU performance counter facility.

Some software resources can be similarly examined. This usually applies to smaller components of software, not to entire applications. For example:

► **Mutex locks.** Utilization may be defined as the time the lock was held, saturation by those threads queued waiting on the lock.

► **Thread pools.** Utilization may be defined as the time threads were busy processing work, saturation by the number of requests waiting to be serviced by the thread pool.

► **Process/thread capacity.** The system may have a limited number of processes or threads whose current usage may be defined as utilization; waiting on allocation may indicate saturation;

and errors occur when the allocation fails (for example, “cannot fork”).

► **File descriptor capacity.** This is similar to the above, but for file descriptors.

If the metrics work well, then use them; otherwise, software troubleshooting can be left to other methodologies.

Suggested Interpretations

The USE method helps identify which metrics to use. After you learn how to read them from the operating system, your next task is to interpret their current values. For some metrics, interpretation may be obvious (and well documented). Others are not so obvious and may depend on workload requirements or expectations. Here are some general suggestions for interpreting metric types:

► **Utilization.** 100% utilization is usually a sign of a bottleneck (check

saturation and its effect to confirm). High utilization (for example, beyond 60%) can begin to be a problem for a couple of reasons. First, when utilization is measured over a relatively long time period (multiple seconds or minutes), a total utilization of, say, 60% can hide short bursts of 100% utilization. Second, some system resources, such as hard disks, usually cannot be interrupted during an operation, even for higher-priority work. Compare this with CPUs, which can be interrupted (“preempted”) at almost any moment. Once disk utilization is above 60%, queueing delays can become more frequent and noticeable, as the tail of the queue becomes longer. This can be quantified using queuing theory to model response time vs. utilization (for example, modeling a disk as M/M/1).

► **Saturation.** Any degree of saturation can be a problem (non-zero). This may be measured as the length of a wait queue or time spent waiting on the queue.

► **Errors.** Non-zero error counters are worth investigating, especially if they are still increasing while performance is poor.

It is easy to interpret the negative cases: low utilization, no saturation, no errors. This is more useful than it sounds. Narrowing the scope of an investigation can help you focus quickly on the problem area.

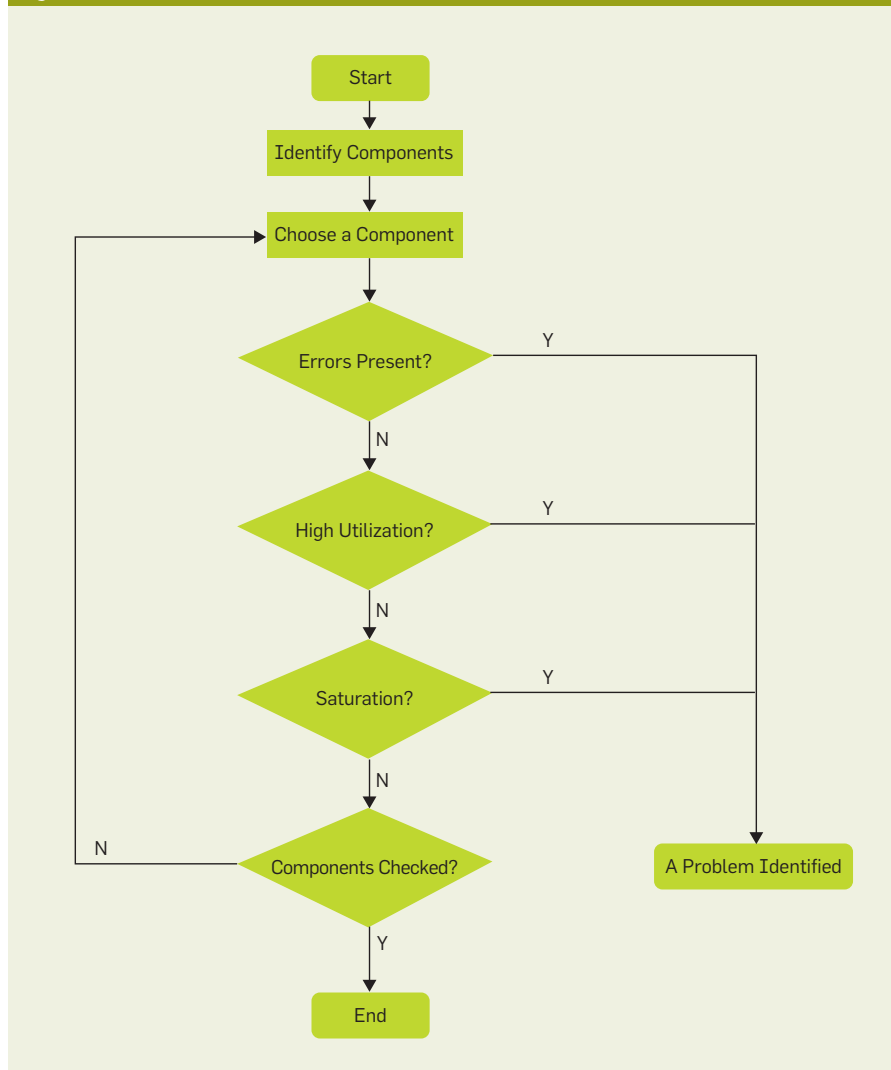
Cloud Computing

In a cloud-computing environment, software resource controls may be in place to limit or throttle tenants who are sharing one system. At Joyent, we primarily use operating-system virtualization (the SmartOS-based SmartMachine), which imposes memory and CPU limits, as well as storage I/O throttling. Each of these resource limits can be examined with the USE method, similar to examining the physical resources.

For example, in our environment *memory capacity utilization* can be the tenant’s memory usage vs. its memory cap. *Memory capacity saturation* can be seen by anonymous paging activity, even though the traditional Unix page scanner may be idle.

► **Strategy.** The USE method is pictured as a flowchart in Figure 2. Errors come first, because they are usually easier and quicker to interpret than utilization and saturation.

Figure 2. The USE method.



The USE method identifies problems that are likely to be system bottlenecks. Unfortunately, a system may be suffering more than one performance problem, so the first issue you find may be a problem but not the problem. You can investigate each discovery using further methodologies before returning to the USE method as needed to iterate over more resources. Or you may find it more efficient to complete the USE method checklist first and list all problems found, then to investigate each based on its likely priority.


Methodologies for further analysis include the workload characterization method and drill-down analysis method, summarized earlier. After completing these (if needed), you should have evidence to determine whether the corrective action needed is to adjust the load applied or to tune the resource itself.

While the previous methodologies may solve most server issues, latency-based methodologies (for example, Method R¹⁴) can approach finding 100% of all issues. These methodologies, however, can take much more time if you are unfamiliar with software internals, and they may be more suited for database administrators or application developers who already have this familiarity.


Conclusion

Systems-performance analysis can be complex, and issues can arise from any component, including from interactions among them. Methodologies in common use today sometimes resemble guesswork: trying familiar tools or posing hypotheses without solid evidence.


The USE Method was developed to address shortcomings in other commonly used methodologies and is a simple strategy for performing a complete check of system health. It considers all resources so as to avoid overlooking issues, and it uses limited metrics so that it can be followed quickly. This is especially important for distributed environments including cloud computing, where many systems may need to be checked. This methodology will, however, find only certain types of issues—bottlenecks and errors—and should be considered as one tool in a larger methodology toolbox.



In a cloud-computing environment, software resource controls may be in place to limit or throttle tenants who are sharing one system.



Acknowledgments

Carry Millsap's methodology research including Method R has been inspirational and my impetus for documenting system methodologies. Thanks to Bryan Cantrill for helping with this article and for fathering DTrace—which has allowed system methodologies to be developed and used in practice far beyond what traditional observability allows. And thanks to Deirdré Straughan for editing and feedback. 

Related articles on queue.acm.org

The Price of Performance

Luiz André Barros

<http://queue.acm.org/detail.cfm?id=1095420>

Performance Anti-Patterns

Bart Smaalders

<http://queue.acm.org/detail.cfm?id=1117403>

Thinking Clearly about Performance

Cary Millsap

<http://queue.acm.org/detail.cfm?id=1854041>

References

1. Allspaw, J. *The Art of Capacity Planning*. O'Reilly, 2008.
2. Cockcroft, A. *Sun Performance and Tuning*. Prentice Hall, NJ, 1995.
3. Function block diagram; http://en.wikipedia.org/wiki/Function_block_diagram.
4. Gregg, B. 7410 hardware update, and analyzing the HyperTransport; <http://dtrace.org/blogs/brendan/2009/09/22/7410-hardware-update-and-analyzing-thehypertransport/>.
5. Gregg, B. The USE method: Linux performance checklist, 2012; <http://dtrace.org/blogs/brendan/2012/03/07/the-use-method-linux-performance-checklist/>.
6. Gregg, B. The USE method: Solaris performance checklist, 2012; <http://dtrace.org/blogs/brendan/2012/03/01/the-use-method-solaris-performance-checklist/>.
7. Gunther, N. *Guerrilla Capacity Planning*. Springer, 2007.
8. Gunther, N. *The Practical Performance Analyst*. McGraw Hill, 1997.
9. Hargreaves, A. I have a performance problem, 2011; <http://alanhargreaves.wordpress.com/2011/06/27/i-have-a-performance-problem/>.
10. Jain, R. *The Art of Computer Systems Performance Analysis*. Wiley, 1991.
11. Loukidas, M. *System Performance Tuning*. O'Reilly, 1990.
12. McDougall, R. and Mauro, J. *Solaris Internals—Solaris 10 and OpenSolaris Kernel Architecture*. Prentice Hall, 2006.
13. McDougall, R., Mauro, J. and Gregg, B. *Solaris Performance and Tools: DTrace and MDB Techniques for Solaris 10 and OpenSolaris*. Prentice Hall, 2006.
14. Millsap, C. and Holt, J. *Optimizing Oracle Performance*. O'Reilly, 2003.
15. Musumeci, G.D. and Loukidas, M. *System Performance Tuning*, 2nd Edition. O'Reilly, 2002.
16. Schlossnagle, T. *Scalable Internet Architectures*. Sams Publishing, 2006.
17. Streetlight effect; http://en.wikipedia.org/wiki/Streetlight_effect.
18. Wong, B. *Configuration and Capacity Planning for Solaris Servers*. Prentice Hall, 1997.

Brendan Gregg is the lead performance engineer at Joyent, where he analyzes performance and scalability at any level of the software stack. He is the primary author of *DTrace* (Prentice Hall, 2011), and co-author of *Solaris Performance and Tools* (Prentice Hall, 2006), and numerous articles about systems performance. He was previously a performance lead and kernel engineer at Sun Microsystems, where he developed the ZFS L2ARC.

Article development led by [acmqueue](http://acmqueue.queue.acm.org)
queue.acm.org

Open source security foundations for mobile and embedded devices.

BY ROBERT N.M. WATSON

A Decade of OS Access-Control Extensibility

TO DISCUSS OPERATING-SYSTEM security is to marvel at the diversity of deployed access-control models: Unix and Windows NT multiuser security, Type Enforcement in SELinux, anti-malware products, app sandboxing in Apple OS X, Apple iOS, and Google Android, and application-facing systems such as Capsicum in FreeBSD. This diversity is the result of a stunning transition from the narrow 1990s Unix and NT status quo to *security localization*—the adaptation of operating-system security models to site-local or product-specific requirements.

This transition was motivated by three changes: the advent of ubiquitous Internet connectivity; a migration from dedicated embedded operating systems to general-purpose ones in search of more sophisticated software stacks; and widespread

movement from multiuser computing toward single-user devices with complex application models. The transition was facilitated by *extensible access-control frameworks*, which allow operating-system kernels to be more easily adapted to new security requirements.

One such extensible kernel reference-monitor framework is the TrustedBSD MAC (Mandatory Access Control) Framework, developed beginning in 2000 and shipped in the open source FreeBSD operating system in 2003. This article first describes the context and challenges for access-control extensibility and high-level framework design, then turns to practical experience deploying security policies in several framework-based products, including FreeBSD, nCircle appliances, Juniper's Junos, and Apple's OS X and iOS. While extensibility was key to each of these projects, they motivated considerable changes to the framework itself, so the article also explores how the framework did (and did not) meet each product's requirements, and finally reflects on the continuing evolution of operating-system security.

A Quiet Revolution in OS Design

Embedded and mobile operating systems have changed greatly in the past 20 years: devices have gained the CPU power to run general-purpose operating systems; they have been placed in ubiquitous networking environments; they have needed to support mature software stacks including third-party applications; and they have found themselves exposed to malicious activity motivated by strong financial incentives. Vendors built on existing operating systems—often open source—to avoid creating them from scratch. This provided mature application frameworks and complex network stacks, both areas of weakness for then-contemporary “embedded operating systems.” One early example is Juniper's Junos, a version of FreeBSD adapted for router control planes in 1998. This trend had come to fruition by 2007 when Google's Android, based



on Linux, and Apple's iOS, based in part on Mach and FreeBSD, became available, transforming the smartphone market.

Common to all of these environments is a focus on security and reliability: as third-party applications are deployed in systems from Junos, via its SDK, and to iOS/Android app stores, sandboxing becomes critical, first to prevent *bricking* (reducing a device to a mere brick as a result of malfunction or abuse) and later to constrain malware. This trend is reinforced by mobile-phone access to online purchasing, and most recently, banking and payment systems. As a result, the role of operating-system security has shifted from protecting multiple users from each other toward protecting a single operator or user from untrustworthy applications. In 2013, embedded devices, mobile phones, and tablets are points of confluence: the interests of many different parties—consumers, phone vendors, application authors, and online services—must be mediated with the help of operating systems

that were designed for another place and time.

Access-Control Frameworks. Operating-system developers must satisfy device vendors, who require everything from router and firewall hardening to mobile-phone app sandboxing. Operating-system vendors had accurately observed a difficult adoption path for historic *trusted operating systems*, whose mandatory access-control schemes suffered from poor usability, performance, maintainability, and—perhaps most critically—end-user demand. Likewise, they saw many promising new security models in research, each with unknown viability, suggesting that no single access-control model would meet all needs. This practical reality of security localization directly motivates extensible access control.

Research over the preceding 20 years had made clear the need for a *reference monitor*—a self-contained, non-bypassable, and compact (hence verifiable) centralization of access control.² By the early 1990s, this concept had been combined with the notion of *en-*

capsulation, appearing in Abrams et al.'s Generalized Framework for Access Control (GFAC),¹ and by the late 1990s in Ott's Rule Set-based Access Control (RSBAC)¹⁴ and Spencer et al.'s Flask security architecture.¹⁷ Mainstream operating-system vendors did not adopt these approaches until the early 2000s with the MAC Framework on FreeBSD²² and shortly after, Linux Security Modules (LSM).²³ In both cases, a key concern was supporting third-party security models without committing to fixed policies as had earlier trusted systems.

The MAC Framework

The MAC Framework was proposed in 1999, with the first whitepaper on its design published in June 2000.²⁰ It appeared in FreeBSD 5.0 in 2003 as an experimental feature—compiled out by default but available to early adopters. FreeBSD 8.0 in 2009 included the framework as a production feature, compiled into the default kernel. (A timeline of key events in its development appears in Figure 1.)

The MAC Framework offers a logical solution to the problem of kernel access-control augmentation: extension infrastructure able to represent many different policies, offering improved maintainability and supported by the operating-system vendor. Similar to device drivers and virtual file system (VFS) modules,¹⁰ policies are

compiled into the kernel or loadable modules and implement well-defined kernel programming interfaces (KPIs). Policies can augment access-control decisions and make use of common infrastructure such as object labeling to avoid direct kernel modification and code duplication. They are able to enforce access control across a broad

range of object types, from files to network interfaces, and integrate with the kernel's concurrency model.

Mandatory Policies. MAC describes a class of security models in which policies constrain the interactions of all system users. Whereas discretionary access control (DAC) schemes such as file-system access-control lists

Figure 1. MAC Framework research and development with key corporate contributions.

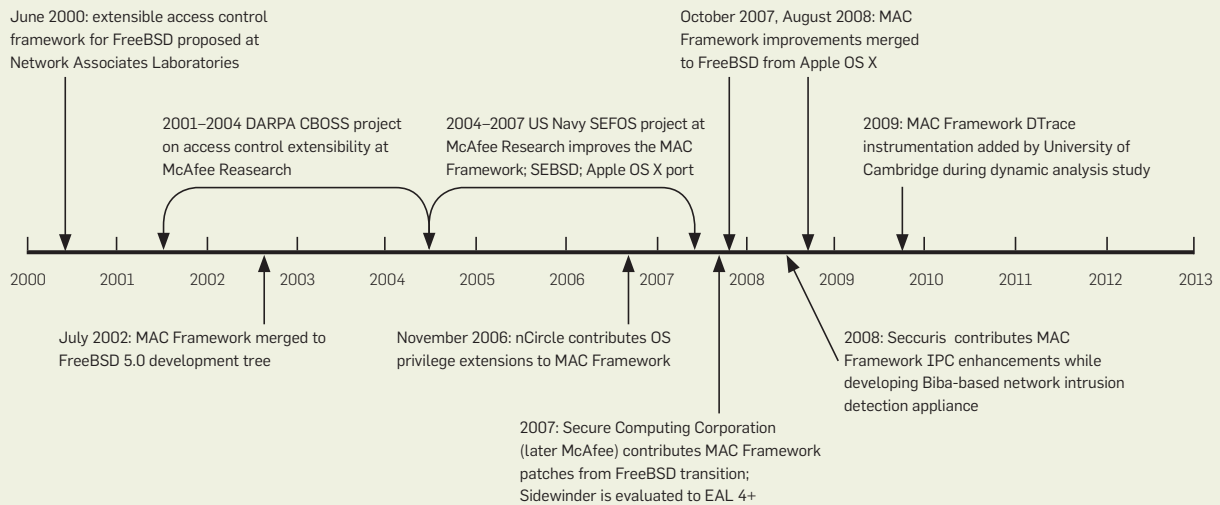
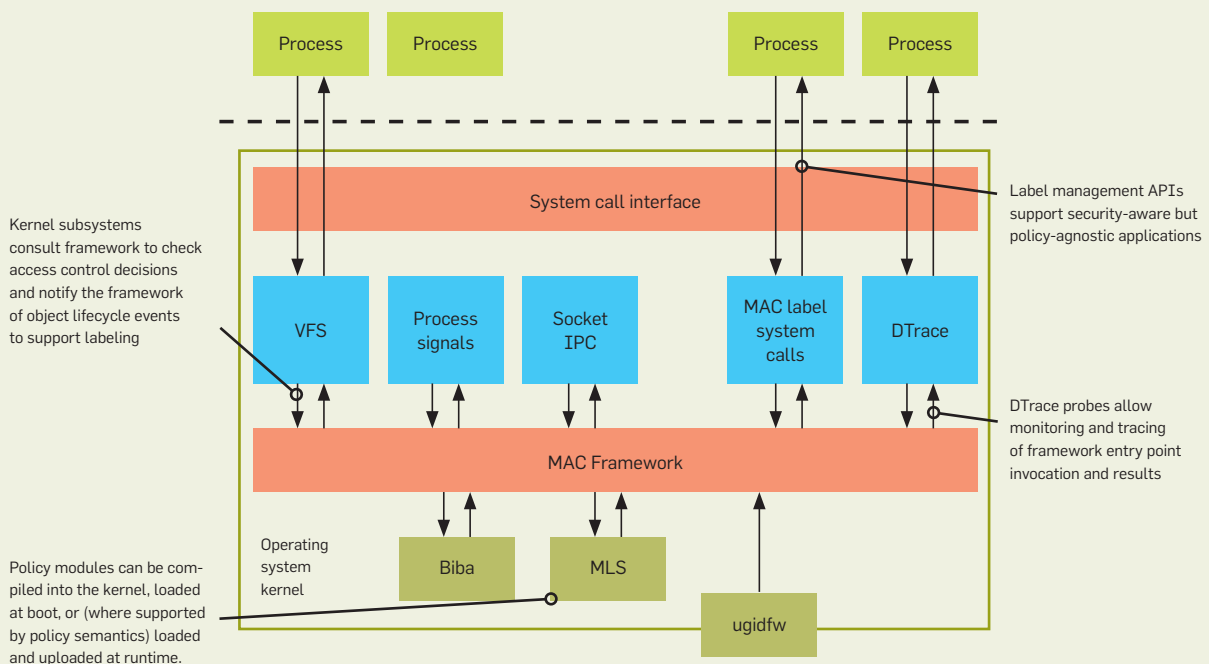


Figure 2. Policy models are encapsulated in kernel modules that augment kernel access control.



(ACLs) allow object owners to protect (or share) objects at their own discretion, MAC enforces systemwide security invariants regardless of user preference. The research literature describes a plethora of mandatory policies grounded in information flow and rule-based models.


Early mandatory policies focused on information flow, requiring ubiquitous enforcement throughout the kernel. *Multilevel security* (MLS) protects confidentiality by labeling user clearance and data confidentiality, limiting flow.⁵ The *Biba integrity policy* is the logical dual of MLS, protecting integrity.⁶ These models maintain subject and object *security labels* holding confidentiality or integrity information, and controlling operations that might lead to information upgrade or downgrade.

SRI International's PSOS (Provably Secure Operating System) design included strong enforcement of object types, supplementing capability protections.¹³ This evolved into Boebert's Type Enforcement (TE)⁷ and Badger et al.'s Domain and Type Enforcement (DTE),⁴ which have proven influential, with TE deployed in SELinux¹¹ and McAfee's Sidewinder firewall. Both models are flexible and fine-grained, labeling subjects and objects with symbolic domains and types. Administrator-controlled rules authorize interactions and transitions between domains.


Finally, a broad class of product-specific *hardening policies* is also relevant; these take less principled approaches, offering direct control over services rather than abstract models.

Before Access-Control Extensibility. In implementation papers, we critiqued contemporaneous techniques from experience:

- ▶ *Direct kernel modification* was used for most trusted systems, whether originated by operating-system vendors (for example, Trusted Solaris) or third-party extensions (for example, Argus Pitbull). Tracking upstream operating-system development is problematic: extensions are unable to depend on public, and hence more stable, APIs (application programming interfaces) and KPIs—and less obvious at the time, ABIs (application binary interfaces) and KBIs (kernel binary interfaces). Upstream churn frequently triggers design and source-



The MAC Framework offers a logical solution to the problem of kernel access-control augmentation: extension infrastructure able to represent many different policies, offering improved maintainability and supported by the operating-system vendor.



code conflicts with security extensions. Assurance is also affected, as the burden of arguing for correctness is left entirely in the hands of the extension writer.

- ▶ *System call interposition* is widely used in antivirus systems and, in the past, security extension products and research systems.⁹ Kernel concurrency proves a particular challenge, and we have demonstrated easily exploited race conditions between wrappers and kernels.¹⁹

Guiding Design Principles. The dual goals of access-control extensibility and encouraging upstream and downstream vendor engagement motivated several design principles for the MAC Framework:

- ▶ *Do not commit to a specific access-control policy.* There is no consensus on a single policy or even policy language; instead, capture policy models in C code.

- ▶ *Avoid policy-specific intrusions into the kernel.* Encapsulate internals behind policy-agnostic interfaces. This leads naturally to object-centered design—access-control checks with respect to subjects, objects, and methods.

- ▶ *Provide policy-agnostic infrastructure.* This satisfies common requirements beyond access-control instrumentation, such as labeling and tracing.

- ▶ *Support multiple simultaneously loaded policies.* In this way different aspects of policy, perhaps from different vendors, can be independently expressed. For example, Trusted IRIX and Argus Pitbull both employed MLS for user-data confidentiality and Biba for trusted computing base (TCB) protection. Composition must be predictable, deterministic, and ideally sensible.

- ▶ *Impose structures that facilitate assurance arguments.* This can be done by separating policy and mechanism via a reference monitor and through well-defined KPI semantics (for example, locking).

- ▶ *Design for an increasingly concurrent kernel.* Policies must not only behave correctly, but also scale with the features they protect.

Architecture of the MAC Framework. The MAC Framework, illustrated in Figure 2, is a thin layer linking kernel services, policies, and security-aware applications. Control passes from kernel consumers to framework to poli-

cies through roughly 250 entry points (object types \times methods):

- *Kernel-service entry points* allow subsystems (for example, VFS) to engage the reference-monitor framework in relevant events and access control.

- *Policy entry points* connect the framework and policies, adding explicit label arguments relative to corresponding kernel-service entry points. They are supplemented by policy life-cycle events and library functions.

Policies need implement only the entry points they require.

- Applications manage labels (on processes and files, among others) using the *label-management API*.

- *DTrace probes* allow entry-point tracing, profiling, and instrumentation.⁸

Collectively, these interfaces allow policies to augment kernel access control in a maintainable manner.

Entry-Point Invocation. To understand how these layers interact, let's follow a single file-write check through the kernel. Figure 3 illustrates `vn_write`, a VFS function implementing the write and `writew` system calls. The `mac_vnode_check_write` kernel service-entry point authorizes a write to a vnode (`vp`) by two subject credentials: `fp->f_cred`, which opened the file, and `active_cred`, which initiated the write operation. Policies can implement Unix *capability semantics* (`fp->f_cred`) or *revocation semantics* (`active_cred`). The vnode lock (`vp->v_lock`) is held over both check and use, protecting label state and preventing time-of-check-to-time-of-use race conditions.

Arguments excluded from entry points are as important as those included. For example, `vn_write`'s data pointer (`uio`) is omitted, as this data resides in user memory and cannot be accessed race-free with respect to the write. Similar design choices throughout the framework discourage behavior not safely expressible through the kernel synchronization model.

Wherever possible, it is best to take the perspective that kernel subsystems implement labeled objects, and that policies may be enforced through controls on method invocation. This approach is a natural fit for the kernel, which adopts an object-oriented structure despite an absence of language features in C. Once objects have been identified, placing entry points requires care: the more granular the KPI, the more expressive policies can be—at the cost of policy complexity. The fewer the calling sites, the easier they are to validate; too few, however, leads to inadequate protection. Entry-point design must also balance placing checks deep enough to allow insight into object types while minimizing enforcement points for a particular level of abstraction.

Figure 3. VFS invokes the MAC Framework to authorize file writes.

```
static int
vn_write(struct file *fp, struct uio *uio,
         struct ucred *active_cred, int flags,
         struct thread *td)
{
    ...
    vn_lock(vp, lock_flags | LK_RETRY);
    ...
#ifdef MAC
    error = mac_vnode_check_write(active_cred, fp->f_cred, vp);
    if (error == 0)
#endif
        error = VOP_WRITE(vp, uio, ioflag, fp->f_cred);
    ...
    VOP_UNLOCK(vp, 0);
    ...
    return (error);
}
```

Figure 4. Framework access control on file writes; lock assertions and DTrace probes are central design elements.

```
int
mac_vnode_check_write(struct ucred *active_cred,
                     struct ucred *file_cred, struct vnode *vp)
{
    int error;

    ASSERT_VOP_LOCKED(vp, "mac_vnode_check_write");
    MAC_POLICY_CHECK(vnode_check_write, active_cred,
                    file_cred, vp, vp->v_label);
    MAC_CHECK_PROBE3(vnode_check_write, error,
                    active_cred, file_cred, vp);
    return (error);
}
```

Figure 5. Biba authorization of file writes.

```
#define LABEL(l) ((struct mac_biba *)mac_label_get((l), biba_slot))

static int
biba_vnode_check_write(struct ucred *active_cred,
                      struct ucred *file_cred, struct vnode *vp, struct label *vplabel)
{
    struct mac_biba *subj, *obj;

    if (!biba_enabled || !revocation_enabled)
        return (0);
    subj = LABEL(active_cred->cr_label);
    obj = LABEL(vplabel);
    if (!biba_dominate_effective(subj, obj))
        return (EACCES);
    return (0);
}
```

Figure 4 illustrates `mac_vnode_check_write`, a thin shim that asserts locks, invokes interested policies, and fires a DTrace probe. Policies are not prohibited from directly accessing `vnode` fields; however, passing an explicit label reference avoids encoding `vnode` structure layout into policies in a common case, improving KPI and KBI resilience.

Policy entry-point invocation, encapsulated in `MAC_POLICY_CHECK`, is nontrivial: access to the policy list must be synchronized to prevent races with module unload, interested policies must be called, and results must be composed. The framework employs a simple composition metapolicy: if any policy returns failure, then access is denied. For example, an `EACCES` returned by Biba would be selected in preference to 0 (success) returned by MLS. The only exception lies in privilege extensions discussed later. This metapolicy is simple, deterministic, predictable by developers, and above all, useful.

Figure 5 illustrates Biba invocation: Biba checks its revocation configuration, unwraps policy-specific labels, and computes a decision using its dominance operator.

Kernel-Object Labeling. Many access-control policies label subjects and objects in order to support access-control decisions (for example, integrity or confidentiality levels). The MAC Framework provides policy-agnostic label facilities for kernel objects, label-management system calls, and persistent storage for file labels. Policies control label semantics—not only the bytes stored, but also the memory model: policies might store per-instance, reference-counted, or global data. For example, when a process creates a new socket, Biba propagates the current subject integrity level (for example, *low*) to the socket label. The partition policy, concerned with inter-process access control, labels only processes and not sockets, so will not assign a label value for the socket.

The framework represents label storage using `struct label`, which is opaque to both kernel services and policies. Where object types support metadata schemes (for example, `mbuf` tags that hold per-packet metadata), those are used; otherwise, label pointers are added to core structures (for example, `vnode`). Policies may borrow existing object locks to protect label

data, where supported by the synchronization model.

From Research to Product

Having presented the design of the MAC Framework, let's turn our attention to policies found in FreeBSD-derived commercial or open source products. Table 1 and Figure 6 illustrate several such policy modules, their feature footprints, and ship dates. A number of factors contributed to the success of this transition:

The need for new access control was pressing. The classic Unix model failed to meet the needs of ISPs, firewalls, and smartphones. Simultaneously, the threat of attack became universal with ubiquitous networking and strong financial incentives for attackers.

Structural arguments for a framework were correct. Access-control extensibility is the preferred way of supporting security localization, catering to diverse requirements.

No one policy model has become dominant. Therefore, many must be supported.

Hardware performance improvement increased tolerance for security overhead. This was true even in consumer and embedded devices.

Table 1. Comparison of policies and their feature footprints.

Name	OSS	CP	Product	Type	Lab	Priv	Proc	VFS	IPC	Net	API	Sig
<code>mac none</code>	✓	-	FreeBSD	Null policy	-	-	-	-	-	-	-	-
<code>mac_stub</code>	✓	-	FreeBSD	Template policy	✓	✓	✓	✓	✓	✓	✓	-
<code>mac_test</code>	✓	-	FreeBSD	Framework self-test	✓	-	✓	✓	✓	✓	✓	-
<code>mac_ugidfw</code>	✓	✓	FreeBSD	File system firewall	-	-	-	✓	-	-	-	-
<code>mac_biba</code>	✓	✓	FreeBSD	Fixed integrity	✓	✓	✓	✓	✓	✓	✓	-
<code>mac_lomac</code>	✓	?	FreeBSD	Floating integrity	✓	✓	✓	✓	✓	✓	✓	-
<code>mac_mls</code>	✓	?	FreeBSD	Confidentiality	✓	-	✓	✓	✓	✓	✓	-
<code>sebsd</code>	✓	✓	FreeBSD	Type Enforcement	✓	✓	✓	✓	✓	✓	✓	-
<code>sandbox</code>	-	✓	Apple OS X	Rule-based	✓	✓	✓	✓	✓	-	✓	✓
<code>quarantine</code>	-	✓	Apple OS X	Taint-based	✓	-	-	✓	-	-	✓	-
<code>tmsafetynet</code>	-	✓	Apple OS X	Fixed integrity	✓	-	-	✓	-	-	✓	-
<code>amfi</code>	-	✓	Apple iOS	Fixed integrity	-	✓	✓	-	-	-	-	✓
<code>sandbox</code>	-	✓	Apple iOS	Rule-based	✓	✓	✓	✓	✓	-	✓	✓
<code>mac_runasnonroot</code>	-	✓	Apple iOS	Hardening	-	-	✓	-	-	-	-	✓
<code>mac_pcap</code>	-	✓	Juniper Junos	Grant BPF privs	✓	✓	-	✓	-	✓	✓	-
<code>mac_veriexec</code>	-	✓	Juniper Junos	Signed binaries	✓	-	-	✓	-	-	-	✓
<code>sidewinder_te</code>	-	✓	McAfee Sidewinder	Type Enforcement	✓	✓	✓	✓	✓	✓	✓	-
<code>mac_ncircle</code>	-	✓	nCircle IP360	Hardening	-	✓	-	✓	-	-	-	-

Key:

OSS: open source software
 CP: shipped in a commercial product
 Lab: uses subject or object label facility

Priv, Proc, VFS, IPC, Net: implements access-control entry points for privileges, processes, file system, interprocess communication, or the network stack

API: uses MAC Framework application APIs
 Sig: provides or depends on application digital signatures

Open source technology transition works. FreeBSD provided not only a forum for collaborative research and development, but also a pipeline to commercial products.

The framework has evolved considerably since 2003 thanks to contributions from companies deploying it in products.

FreeBSD

FreeBSD is an open source operating system used to build online services, appliances, and embedded devices. FreeBSD or its components can be found in data centers (Internet Systems Consortium, Yahoo!), as a foundation for integrated products (NetApp and EMC Isilon storage appliances), and in embedded/mobile devices (Juniper switches and Apple iPhones). Its origins lie in BSD (the Berkeley Software Distribution), developed in the 1970s and 1980s.¹² BSD originated a number of central Unix technologies, including FFS (the Fast File System) and the Berkeley TCP/IP stack and sockets API. The BSD license and its variations (MIT, CMU, ISC, Apache) have encouraged technology transition by allowing unrestricted commercial use. FreeBSD's diverse consumers both motivate and are the perfect target for security localization.

The MAC Framework is a complex piece of software; although the framework itself is only 8,500 lines of

code, with 15,000 lines in reference policies, it integrates with a multi-million-line kernel. The transition to production relied on several factors, including increasing confidence in mediation and response to community feedback on design, compatibility, and performance. The framework, as first shipped in FreeBSD 5.0, was marked as *experimental*, with several implications:

- ▶ Enabling it required recompiling the kernel.

- ▶ Documentation marked it as potentially incomplete, unstable, or insecure, and therefore unsupported.

- ▶ Programming and binary interface (API, KPI, ABI, and KBI) stability was disclaimed, allowing change without formal depreciation.

Merging the framework while still experimental was key to gaining users who could help validate and improve the approach, while retaining the flexibility to make changes. Two concerns needed to be addressed before the framework could be considered production worthy:

- ▶ Binary compatibility impact for the kernel, policies, and other modules must be better understood.

- ▶ Performance must be analyzed and optimized based on community review.

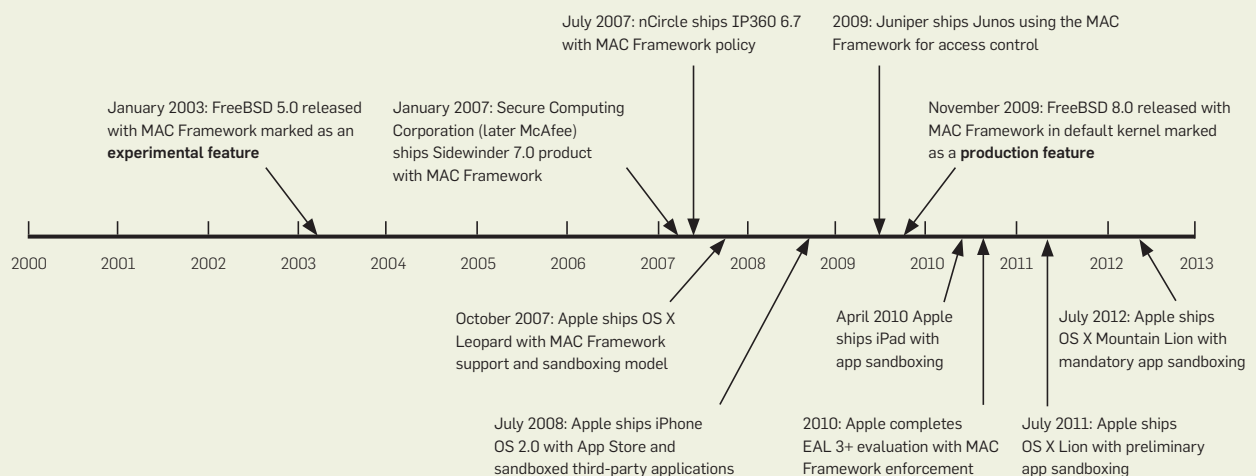
KPI and KBI Resilience. FreeBSD policy dictates that certain classes of kernel modules compiled against a release must work with later minor

versions in the same series (for example, a FreeBSD 9.0 network device driver should work with FreeBSD 9.1). The goals were to avoid disrupting the KBIs of consumer subsystems and to offer similar levels of binary compatibility for policy modules. Label storage opacity for subsystems and policies was the primary area of refinement, which avoids encoding kernel data-structure internals into policies if they require only label access, as well as providing flexibility to change label implementation.

Performance Optimization. Many FreeBSD deployments are extremely performance sensitive, requiring minimal overhead, especially if the framework is disabled. As sites select policies based on local security-performance trade-offs, it is also desirable for policies to incur only the performance penalties of features they actually use—*performance proportionality*. As shipped in FreeBSD 5.0, however, regressions were measurable, an obstacle to enabling the framework by default.

Label Allocation Trade-offs. Even when the framework was compiled out, bloat from adding a label to kernel data structures (especially packet mbufs) created significant allocation-time zeroing cost. In FreeBSD 5.1, in-lined mbuf labels were replaced with pointers, and for all object types in 5.2; this decreased costs for non-MAC

Figure 6. Timelines of selected MAC Framework-based product ship dates.



kernels at the expense of additional allocation and indirection for MAC-enabled kernels.

Label allocation was even more measurable with the framework enabled—and unnecessary for unlabeled policies. The effect was most pronounced with network packets and led, in FreeBSD 5.1, to a per-policy flag to request packet labels. In 8.0, this approach was generalized so that labels were allocated only for object types for which at least one loaded policy defined an initialization entry point. This effectively eliminated the cost of labeling when not required by a policy, restoring performance proportionality and satisfying the general case well. However, one commercial product that used packet labeling, the McAfee Sidewinder Firewall, saw sufficient overhead to bypass the label abstraction in favor of direct structure modification.


Minimizing Synchronization Overheads. With the framework compiled in, lock-protected reference count operations on entry-point invocation were easily measurable for frequent operations, such as per-packet delivery checks. As multicore hardware became more common, lock (and later cache-line) contention also became significant.

Beginning in FreeBSD 5.2, policies were divided into *static* and *dynamic* sets to help fixed-configuration embedded systems. The former were compiled in or loaded at boot and unloadable thereafter, and hence required no synchronization. Dynamic policies—those loaded after boot, or potentially unloadable—still required multiple lock operations.


In FreeBSD 8.0, synchronization was further optimized so that the MAC Framework could be shipped in the default kernel. This effort benefited from continuing improvements in kernel scalability driven by increasingly common eight-core machines. Particularly critical were *read-mostly locks*, which do not trigger cache-line migrations during read-only acquisition, at the cost of more expensive exclusive acquisition—perfect for infrequently changed policy lists.

nCircle IP360 Appliance

nCircle Network Security produces a FreeBSD-based appliance, the IP360,



It is desirable for policies to incur only the performance penalties of features they actually use—performance proportionality.



to scan networks for vulnerable software and Sarbanes-Oxley compliance. While most of its security requirements could be captured with conventional DAC, customers requested the ability to audit appliance content and configuration directly. To meet this requirement, while limiting potential damage in case audit access is misused or compromised, nCircle developed a custom policy.

The policy authorizes an *audit user* to read all file-system and configuration data, bypassing permissions, while also preventing file-system writes. The MAC Framework could express only a subset of this augmentation: policies could constrain rights but not grant them. nCircle therefore enhanced the framework to allow control over fine-grained *system privileges*.

Privilege Extensions. Operating-system privilege confers the right to bypass operating-system security policies (for example, changing system settings or overriding DAC or the process model). In classic Unix, system privileges are granted to any process running as the root user. To meet nCircle's goals, a policy must be able to augment the kernel's default privilege policy to grant (and moderate) privileges for other users. This presented two technical challenges: how to identify and distinguish different types of privilege; and how to add extensibility to the existing privilege model. These problems resemble, in microcosm, the larger concern addressed by the MAC Framework—structuring of a reference monitor for extensibility—and seemed a natural fit despite a departure from the original design choice to only limit, rather than grant, rights.

All existing kernel privilege checks were analyzed and replaced with checks for specific named privileges. Privilege checking was then reworked to include an explicit composition policy for sources and limitations of privilege, including two new MAC Framework entry points: `mac_priv_check` follows the standard entry-point conventions, accepting a credential, named privilege arguments, and restricting privileges by returning an error; `mac_priv_grant` diverges from this model by overriding the base operating-system policy to grant new rights, using a new composition operator that allows any policy to grant a right, rather

Table 2. Apple OS X applications may use one of several statically configured profiles, or define their own.

Profile	Description
kSBXProfileNoInternet	TCP/IP networking is prohibited
kSBXProfileNoNetwork	All sockets-based networking is prohibited
kSBXProfileNoWrite	File-system writes are prohibited
kSBXProfileNoWriteExceptTemporary	File-system writes are restricted to temporary folders
kSBXProfilePureComputation	Only Mach IPC to the host process is permitted

than requiring them all to agree.

Existing policies were updated to take advantage of the new features, providing stronger nondiscretionary control of the root user. For example, the Biba policy now limits access to a number of privileges that might allow bypass of the process model or system reconfiguration when operating as the root user without Biba privilege. These features shipped in FreeBSD 7.0.

The nCircle MAC Policy. The nCircle policy extends (and restricts) rights available to the *audit user*:

- ▶ It identifies a specific user ID to which all remaining policy activities apply.

- ▶ Privileges are granted, including read access to the kernel log and firewall configuration, and file read/lookup protections are overridden.

- ▶ VFS entry points deny write access to all objects and read access to certain files such as the password file.

With these enhancements, the nCircle policy is able to combine controlled privilege escalation with mandatory constraints, meeting product needs while minimizing local operating-system modification.

Juniper Junos

The Junos router operating system runs on the control planes of all Juniper routers and switches. Juniper maintains substantial local modifications to FreeBSD and is undergoing a multiyear process to minimize its patches by returning improvements to the FreeBSD community and increasing use of operating-system extensibility frameworks that allow local features to be cleanly grafted onto an unmodified operating system. As part of that project, Juniper has been moving local security extensions into MAC Framework policies, both to reduce conflicts during FreeBSD updates and to prepare certain policies

for upstreaming. Junos ships with four local security extensions:

- ▶ `mac_runasnonroot`. Ensures that third-party applications written against the Junos SDK are not run as the root user.

- ▶ `mac_pcap`. Allows Junos SDK applications to capture packets despite not running as root.

- ▶ `mac_veriexec`. Implements support for digitally signed binaries.

- ▶ Junos SDK sandboxing. Constrains third-party applications based on `mac_veriexec` certificates.

The `mac_runasnonroot` and `mac_pcap` extensions first shipped as framework policies in 2009. Then `mac_veriexec` shipped in 2012, replacing a previous directly patched implementation. Juniper is preparing to migrate Junos SDK sandboxing to the MAC Framework to reduce local patches further, as well as upstream `mac_veriexec`.

These policies required minor changes to the MAC Framework, including additional entry points; perhaps most interesting is a new `O_VERIFY` flag to the `open` system call, which signals to the framework that the user-space runtime linker has requested that a file be validated.

Apple OS X and iOS

In quick succession, Apple released versions of OS X Leopard for the desktop/server in 2007, and iPhone OS 2 for the iPhone and iPod Touch in 2008, incorporating the MAC Framework as a reference-monitor framework. OS X Snow Leopard shipped with three MAC policies:

- ▶ **Sandbox.** Provides policy-driven sandboxing of risky components that process untrustworthy data such as network services and video codecs.

- ▶ **Quarantine.** Taints downloaded files, supporting a user dialog display-

ing the originating website.

- ▶ **Time Machine Safety Net.** Protects the integrity of Time Machine backups.

With OS X Mountain Lion, applications distributed via Apple's App Store have mandatory sandboxing. Apple's iOS 2.0 shipped with two policies: Sandbox and one additional:

- ▶ **Apple Mobile File Integrity (AMFI).** Works in concert with a code-signing facility, terminating apps whose digital signatures have been invalidated at runtime; exempts debugging during app development.

Collectively the policies support system integrity and provide strong separation between apps in order to keep data private. Both OS X and iOS diverge substantially from our design expectations for the MAC Framework, requiring significant adaptation.

XNU Prototype. Apple began beta testing OS X in 2000, and the promise of a commodity desktop operating system with an open source kernel was difficult to ignore. The XNU kernel is a sophisticated blend of Carnegie Mellon University's Mach microkernel, FreeBSD 5.0, cherry-picked newer FreeBSD elements, and numerous features developed by Apple. With these foundations, it seemed likely that the MAC Framework approach, and even code, would be reusable.

Though not a microkernel, XNU (short for X is not Unix) adopts many elements from Mach, including its scheduler, interprocess communication (IPC) model, and VM system. The FreeBSD process model, IPC, network stack, and VFS are grafted onto Mach, providing a rich POSIX programming model. Apple-developed kernel components in the first release of OS X included the I/O Kit device-driver framework, network kernel extensions (NKEs), and the HFS+ file system; this list has only grown over time.

Interesting questions abounded: for example, would ideas developed in the DTOS¹⁶ and DTOS¹⁷ microkernel projects apply better or worse than the monolithic kernel approach in the MAC Framework? Between 2003 and 2007, the increasingly mature MAC Framework was ported to OS X.¹⁸

Adapting to OS X. The MAC Framework required a detailed analysis of the FreeBSD kernel and is tightly integrated with low-level memory management and synchronization, as well as higher-level services such as the file system, IPC, and network stack. While the adaptation to OS X was able to rely heavily on Apple's use of FreeBSD components, fundamental changes were needed to reflect differences between FreeBSD and XNU.

The first step was integrating the MAC Framework with the closely aligned BSD process model, file system, and network stack. High-level architectural alignment made some of the adaptation easy, but nontrivial differences were also encountered. For example, FreeBSD's Unix file system (UFS) considers directories to be specialized file objects, whereas HFS+ considers the directory and object attribute structure, or *disk catalog*, to be a first-class object. This required changes to both the framework and XNU.

Next, coverage was extended to include Mach tasks and IPC. Each XNU process links a Mach task (scheduling, VM) with a FreeBSD process (credentials, file descriptors), presenting a philosophical problem: is the MAC Framework part of Mach or BSD? While useful architecturally, the Mach-BSD boundary in XNU proves artificial: references frequently span layers, requiring the MAC Framework to serve both. Label modifications on BSD process labels are mirrored to corresponding Mach task labels.

Mach ports are another case in which microkernel origins come into conflict with the monolithic kernel premise of the MAC Framework. Unlike BSD IPC objects, with kernel-managed namespaces, Mach ports rely on userspace namespaces managed by *launchd* (for example, for desktop IPC). Taking a leaf from DTOS, *launchd* is responsible for labeling and enforcement but queries the reference monitor to authorize lookups.

A userspace *label handle* abstraction similar to the kernel *label* structure serves this purpose.

Adoption by Apple. Apple is the world's largest vendor of desktop Unix systems and was among the first to deploy Unix in a smartphone. It has likewise seen exploding use cases and new security requirements motivated by ubiquitous networking and malicious attackers. Apple's adoption of the MAC Framework was not assured, however, as competing technologies were also considered, motivated by similar observations, awareness of future product directions, performance concerns, and our research.

Alternatives included system-call interposition-based technology similar to that discussed earlier, and Apple's Kauth³ (short for kernel authorization), an authorization framework targeted at antivirus vendors (modeled in part on the MAC Framework). Apple found arguments about the fallibility of system-call interposition convincing, and in the end adopted two technologies: Kauth for third-party antivirus vendors; and the more expressive and capable MAC Framework for its own sandboxing technologies.

The Sandbox Policy. Since Apple's OS X and iOS policy modules are not open source, we are unable to consider their implementations, but public documentation exists for the Sandbox policy used by Mac OS X components

and third-party applications such as Google's Chrome Web browser. Sandbox allows applications voluntarily to restrict their access to resources (for example, the file system, IPC namespaces, and networking). Process sandbox profiles are stored in process labels.

Bytecode-compiled policies can be set via public APIs, or by the *sandbox-exec* helper program. Applications may select from several Apple-defined policies (Table 2) or define custom policies. Several applications use default policies such as the iChat video codec, which employs the computation-only profile limited to IPC with the host process. Many other software components, such as Spotlight indexing, the BIND name server, Quicklook document previews, and the System Log Daemon, utilize custom profiles to limit the effects of potential vulnerabilities.

Figure 7 shows excerpts from the *common.sb* profile used by Chrome, illustrating key Sandbox constructs: coarse controls for *sysctl* kernel-management interfaces and shared memory, and fine-grained regular expression matching of file paths. File path-based control is a highlight of the Sandbox policy, addressing programmer models much better than file labels in Biba, MLS, and TE. Path-based schemes are difficult to implement on the Unix VFS model, which considers paths to be second-class constructs. Whereas FreeBSD permits files to have zero (unlinked

Figure 7. Chrome OS X sandbox policy excerpts.

```
(deny default)

; Allow sending signals to self - http://crbug.com/20370
(allow signal (target self))

; Needed for full-page-zoomed controls -
; http://crbug.com/11325
(allow sysctl-read)

; Allow following symlinks
(allow file-read-metadata)

; Loading System Libraries.
(allow file-read-data
 (regex #"^/System/Library/Frameworks($|/)" ))
(allow file-read-data
 (regex #"^/System/Library/PrivateFrameworks($|/)" ))
(allow file-read-data
 (regex #"^/System/Library/CoreServices($|/)" ))

; Needed for IPC on 10.6
(allow ipc-posix-shm)
```

but open), one, or multiple names (hard links), HFS+ implements a parent pointer for files and ensures that the name cache always contains the information required to calculate unambiguous paths for in-use files.


While Sandbox is used with many OS X services, a number of third-party applications incorporate strong assumptions of *ambient authority*, the ability to access any object in the system. With the iPhone, Apple broke this assumption: applications execute in isolation from system services and each other. This model is now appearing in OS X and could similarly help protect device integrity against misbehaving apps and, increasingly, end-user data.

Performance Optimizations. OS X and iOS were shipped with the MAC Framework prior to FreeBSD 8.0's performance optimizations, requiring Apple to make its own optimizations based on product-specific constraints. As with FreeBSD optimizations, these were generally concerned with the overhead of framework entry and labeling. By default, labeling is compiled out of the kernel for certain object types; for others, such as *vnodes*, policies may selectively request label allocation, catering to the often-sparse labeling use in OS X's policies.


In FreeBSD, framework instrumentation and synchronization optimizations rely on all-or-nothing distinctions between sites willing to pay for additional access-control extension. In OS X, the assumption is that sandboxing is used on most machines, but selectively applied to high-risk processes. To this end, each process carries a mask, set by policies, indicating which object types require enforcement. As OS X adopts more universal sandboxing, as is the case in iOS, it may be desirable to apply more global optimizations as in FreeBSD.

Reflections

Over the past decade, the MAC Framework has become the foundation for numerous instances of security localization, allowing local access-control policies to be composed with the still-popular Unix discretionary access control (DAC) model—a timely convergence of industry requirements and research. Deploying via open source proved a successful strategy, providing



The MAC Framework has become the foundation for numerous instances of security localization, allowing local access-control policies to be composed with the still-popular Unix discretionary access control model.



a forum for collaborative refinement, access to early adopters, and a path to numerous products.

Perhaps the most surprising adoption was at McAfee itself: when the framework was open sourced by McAfee Research, Secure Computing Corporation (then a competitor) adopted it for Sidewinder, which McAfee later acquired. More generally, this speaks to the success of open source in providing a venue in which competing companies can collaborate to develop common infrastructure technologies. The industry's adoption of open source foundations for appliances and embedded devices has been well-catered to by our access-control extensibility argument:

- ▶ Security localization in devices has become widespread.
- ▶ The criticality of multiprocessing has only increased.
- ▶ Security label abstractions have proven beneficial beyond their MAC roots.
- ▶ Non-consensus on access-control policies continues.

The MAC Framework, however, also required refinement and extension to address several unanticipated concerns:

- ▶ The desire to revisit the structure of Unix privilege.
- ▶ The importance of digital signatures when applying access control to third-party applications.
- ▶ Continued tensions over the desire for name-based vs. label-based access control.

New Design Principles. In light of extensive field experience with the MAC Framework, we have added several new design principles:

Policy authors determine their own performance, functionality, and assurance trade-offs. Policies may not require heavyweight infrastructure (for example, labels), so offer performance proportionality.

Traceability is a key design concern.

Programming and binary interface stability is critical. API, ABI, KPI, and KBI sustainability is often overlooked in research, where prototypes are frequently one-offs without multi-decade support obligations.

Manipulating operating-system privilege is important to policies that augment rather than supplement DAC.

It is clear from the work of down-

stream consumers, however, that two further principles are now also necessary:

Application authors are first-class principals. Apple's App Store and Juniper's SDK both employ application signatures and certificates as policy inputs.

Applications themselves require flexible access control to support application compartmentalization.

This latter observation led us to develop the application-focused Capsicum protection model,²¹ recently shipped as an experimental feature in FreeBSD 9.0. It can be viewed as complementary to policy-driven kernel access control.

Domain-Specific Policy Models.

Why no consensus has been reached in the expression of operating-system policies is an interesting question—certainly, proponents of successive policy models have argued that their models capture the key concerns in system design. In catering to a variety of models, our observations are twofold: first, policy models aim to capture aspects of the *principle of least privilege*¹⁵ but often in fundamentally different forms (for example, information flow vs. system privileges), making their approaches complementary; second, different models address different spaces in a multidimensional trade-off between types of expression, assurance, performance, administrative complexity, implementation complexity, compatibility, and maintainability. This instead reflects a consensus for *domain-specific policy models*.

The Value of Extensibility. Does the need for significant design enhancement confirm or reject the hypothesis of access-control extensibility? Further comparison to similar frameworks, such as VFS and device drivers, seems appropriate: both are regularly extended to adapt to new requirements such as changes in distributed file-system technology or improvement in power management. The willingness of industrial consumers to extend the framework and return improvements reflects our fundamentally economic hypothesis regarding extensibility: managing the upstream-downstream relationship for significant source-code bases is a strong motivator. Widespread and continuing

deployment of the MAC Framework appears to confirm the more general argument that access-control extensibility is a critical aspect of contemporary operating-system design.

Acknowledgments

Systems research emphasizes the practical application of ideas to real-world systems: only by implementing an idea can you fully understand it; this is even more true in the transition from research to practice. Acknowledgments are due to a large cast spanning many institutions; to view full list, see queue.acm.org. Helpful feedback on this article came from Ross Anderson, Simon Cooper, Jon Crowcroft, Simon Gerraty, Matthew Grosvenor, Steve Hand, Mark Handley, Steve Kiernan, Anil Madhavapeddy, Peter G. Neumann, George Neville-Neil, and Mike Silbersack.

This work was supported by DARPA/AFRL contract FA8750-10-C-0237 (CTSRD), with previous support from the DARPA CBOSS and SPAWAR SEFOS contracts, spanning the DARPA CHATS and DARPA CRASH research programs. The views, opinions, and/or findings in this article are author's and should not be interpreted as representing the official views or policies, either expressed or implied, of DARPA, the U.S. Navy, AFRL, or the Department of Defense. Google, Inc. also supported this work. □

Related articles on queue.acm.org

Building Systems to Be Shared, Securely

Poul-Henning Kamp, Robert Watson
<http://queue.acm.org/detail.cfm?id=1017001>

Extensible Programming for the 21st Century

Gregory V. Wilson
<http://queue.acm.org/detail.cfm?id=1039534>

ACM CTO Roundtable on Mobile Devices in the Enterprise

Mache Creeger
<http://queue.acm.org/detail.cfm?id=2016038>

References

1. Abrams, M.D., Eggers, K.W., LaPadula, L.J. and Olson, I.M. A generalized framework for access control: An informal description. In *Proceedings of the 13th NIST-NCSC National Computer Security Conference* (1990), 135–143.
2. Anderson, J.P. Computer Security Technology Planning Study. Technical report, Electronic Systems Division, Air Force Systems Command, 1972.
3. Apple Inc. Kernel authorization. Technical Note TN2127, 2007; <http://developer.apple.com/technotes/tn2005/tn2127.html>.

4. Badger, L., Sterne, D.F., Sherman, D., Walker, K.M. and Haghghat, S.A. Practical domain and type enforcement for Unix. In *Proceedings of the 1995 IEEE Symposium on Security and Privacy* 66 (1995). IEEE Computer Society.
5. Bell, D.E., and L.J. LaPadula. Secure computer systems: mathematical foundations and model. Technical Report M74-244, Mitre Corp., Bedford, MA, 1973.
6. Biba, K. Integrity considerations for secure computer systems. Technical Report TR-3153, Mitre Corp., Bedford, MA, 1977.
7. Boebert, W.E. and Kain, R.Y. A practical alternative to hierarchical integrity policies. In *Proceedings of the 8th National Computer Security Conference*, 1985.
8. Cantrill, B.M., Shapiro, M.W. and Leventhal, A.H. Dynamic instrumentation of production systems. In *Proceedings of the Usenix Annual Technical Conference* (Berkeley, CA, 2004). Usenix Association.
9. Fraser, T., Badger, L. and Feldman, M. Hardening COTS software with generic software wrappers. In *Proceedings of the 1999 IEEE Symposium on Security and Privacy*.
10. Kleiman, S.R. Vnodes: An architecture for multiple file system types in Sun Unix. In *Proceedings of the Summer 1986 Usenix Conference*.
11. Loscocco, P.A. and Smalley, S.D. Integrating flexible support for security policies into the Linux operating system. In *Proceedings of the 2001 Usenix Annual Technical Conference*. Usenix Association, 29–42.
12. McKusick, M.K., Neville-Neil, G.V. *The Design and Implementation of the FreeBSD Operating System*. Pearson Education, 2004.
13. Neumann, P.G., Boyer, R.S., Feiertag, R.J., Levitt, K.N. and Robinson, L. A provably secure operating system: the system, its applications, and proofs, second edition. Technical Report CSL-116, Computer Science Laboratory, SRI International, 1980.
14. Ott, A. Rule-set-based access control (RSBAC) for Linux (2010); <http://www.rsbac.org/>.
15. Saltzer, J.H. and Schroeder, M.D. The protection of information in computer systems. In *Proceedings of the IEEE* 63, 9 (1975), 1278–1308.
16. Sebes, E.J. Overview of the architecture of Distributed Trusted Mach. In *Proceedings of the Usenix Mach Symposium* (1991). Usenix Association, 20–22.
17. Spencer, R., Smalley, S., Loscocco, P., Hibler, M., Andersen, D. and Lepreau, J. 1999. The Flask security architecture: system support for diverse security policies. In *Proceedings of the 8th Usenix Security Symposium* (1999). Usenix Association, 123–139.
18. Vance, C., Miller, T. C., Dekelbaum, R., Reisse, A. 2007. Security-enhanced Darwin: Porting SELinux to Mac OS X. In *Proceedings from the Third Annual Security Enhanced Linux Symposium* (2007).
19. Watson, R.N.M. Exploiting concurrency vulnerabilities in system call wrappers. In *Proceedings of the First Usenix Workshop on Offensive Technologies*. Usenix Association, 2007, 1–8.
20. Watson, R.N.M. New approaches to operating system security extensibility. Technical Report UCAM-CL-TR-818, University of Cambridge, Computer Laboratory, 2012.
21. Watson, R.N.M., Anderson, J., Laurie, B. and Kennaway, K. Capsicum: Practical capabilities for Unix. In *Proceedings of the 19th Usenix Security Symposium* (2010). Usenix Association.
22. Watson, R.N.M., Feldman, B., Migus, A. and Vance, C. Design and implementation of the TrustedBSD MAC Framework. In *Proceedings of the Third DARPA Information Survivability Conference and Exhibition* (2003). IEEE.
23. Wright, C., Cowan, C., Morris, J., Smalley, S. and Kroah-Hartman, G. 2002. Linux security modules: General security support for the Linux kernel. In *Proceedings of the 11th Usenix Security Symposium* (2002). Usenix Association.

Robert N.M. Watson is a senior research associate at the University of Cambridge Computer Laboratory, and a Research Fellow at St John's College Cambridge. He was a senior principal scientist at SPARTA ISSO, and a senior research scientist at McAfee Research, where he led the development of a kernel access control extension framework for the open source FreeBSD operating system, now used in products such as Junos, Apple OS X, and iOS.

DOI:10.1145/2408776.2408793

Extending the data trust perimeter from the enterprise to the public cloud requires more than encryption.

BY ARI JUELS AND ALINA OPREA

New Approaches to Security and Availability for Cloud Data

CLOUD COMPUTING IS a service model that offers users (herein called “tenants”) on-demand network access to a large shared pool of computing resources (the cloud). The financial benefits of cloud computing are widely recognized. Building and managing a large-scale data center results in savings of a factor between five and seven over one of medium size in terms of electricity, hardware, network-bandwidth, and operational costs.¹ From the tenant’s perspective, the ability to use and pay for resources on demand and the elasticity of the cloud are strong incentives for migration to the cloud.

Despite these benefits, public clouds are still not widely adopted, especially by enterprises. Most large organizations today run private clouds, in the sense

of virtualized and geographically distributed data centers, but rarely rely primarily on externally managed resources; notable exceptions include Twitter and *The New York Times*, which run on Amazon infrastructure.

Major barriers to adoption are the security and operational risks to which existing cloud infrastructures are prone, including hardware failure, software bugs, power outages, server misconfiguration, malware, and insider threats. Such failure and attack vectors are not new, but their risk is amplified by the large scale of the cloud.⁵ They can even be disastrous, including data loss and corruption, breaches of data confidentiality, and malicious tampering with data. Strong protections beyond encryption are therefore a necessity for data outsourced to the cloud, with two standing out as particularly important: integrity, or assurance against data tampering, and freshness, or the guarantee that retrieved data reflects the latest updates.

Another concern hindering migration into a public cloud is lack of availability and reliability guarantees. Well-known cloud providers have experienced temporary lack of availability lasting at least several hours^{11,21} and striking loss of personal customer data, most notably the 2011 T-Mobile/Sidekick incident.²³ Traditional reliability models for hardware make certain assumptions about failure patterns (such as independence of failures among hard drives) that are not accurate in the world of cloud computing. Without novel data-reliability protections

» key insights

- **Security concerns limit or even impede enterprise migration into public clouds.**
- **Creating incentives for cloud adoption requires thinking beyond data encryption, which alone rarely provides confidentiality on data processed in the cloud or protects against tampering, corruption, or loss of availability.**
- **Real-time auditing by tenants or third parties can create new security visibility in the cloud and strong assurance of correct cloud-service operation.**



(beyond today's RAID-5 and RAID-6), maintaining correctness of massive amounts of data over long periods of time is extremely difficult.⁴

Another top concern for enterprises migrating into the cloud is collocation with potentially malicious tenants.⁵ In an Infrastructure-as-a-Service (IaaS) model, tenants rent virtual machines (VMs) on servers they share with other tenants; logical isolation among VMs is enforced by hypervisors. In a Platform-as-a-Service (PaaS) model, different tenants may run applications in the same operating system, without clear isolation beyond basic OS-level protections (easily bypassed by advanced malware). Ristenpart et al.¹⁸ showed that an attacker can collocate a VM under its control on the same server as a targeted victim VM in the Amazon IaaS infrastructure; they also provided

evidence that such an attacker can exploit side channels in shared hardware (such as the L2 cache) to exfiltrate sensitive data from the victim.

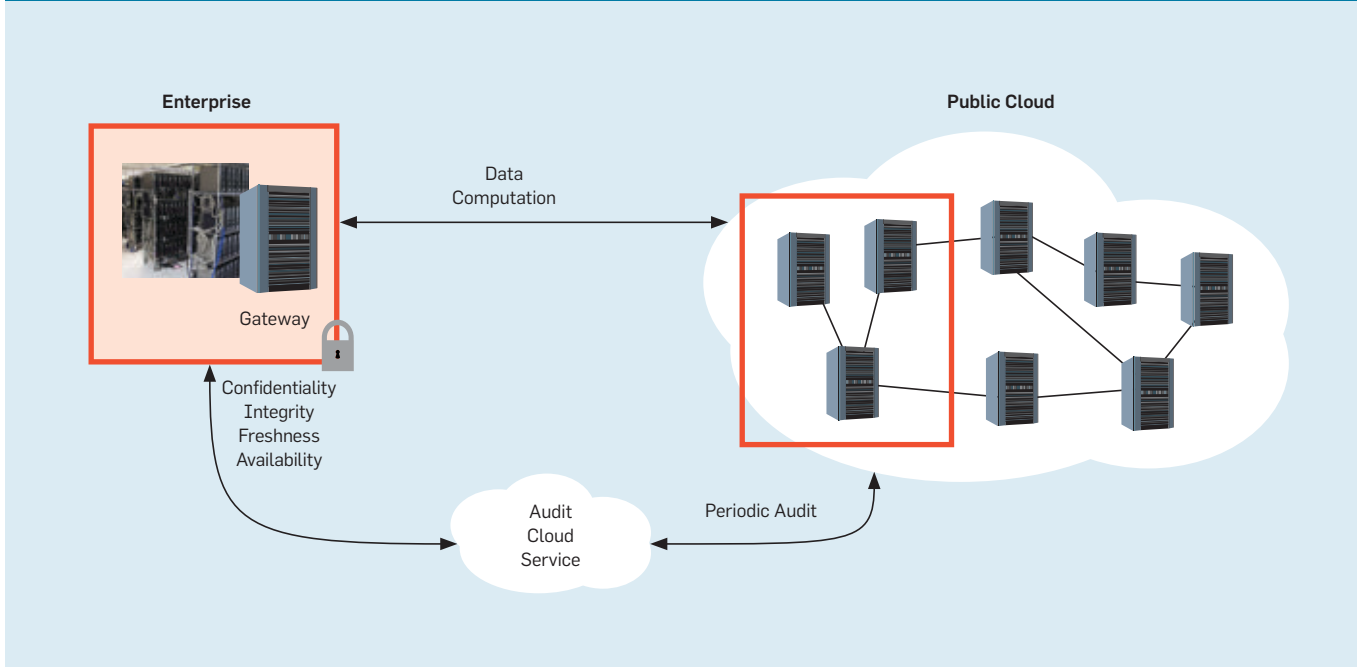
Our research addresses the challenge of migrating enterprise data into the public cloud while retaining tenant trust and visibility. We have devised cryptographic protocols that extend traditional trust perimeters from enterprise data centers into the public cloud by conferring strong protections on migrated data, including integrity, freshness, and high availability. In addition, we propose an auditing framework to verify properties of the internal operation of the cloud and assure enterprises that their cloud data—and workloads—are handled securely and reliably.

Adversarial model. Here, we are concerned mainly with cloud provid-

ers subject to a range of threats. These providers might deviate from our protocols for cost savings or due to poor security practices but not for pure malicious intent. In most cases, we are able to detect deviations from our protocols by misbehaving cloud providers but do not provide remediation mechanisms against fully malicious cloud providers. By using multiple cloud providers in the design of our High-Availability and Integrity Layer, or HAIL, protocol, described later, we show we can also provide data integrity and availability in a setting in which a fraction of providers can be fully compromised.

Solution overview. Figure 1 outlines our vision of a more-trustworthy cloud-computing model for enterprises. A small trusted gateway within the enterprise intermediates all communication, from internal data

Figure 1. Extending trust perimeter from enterprise data center to the public cloud.



center to external public cloud. The gateway manages cryptographic keys (for encrypting data for confidentiality requirements), maintains trusted storage for integrity and freshness enforcement, and may add redundancy to data for enhanced availability. Once the data and workloads of a particular enterprise migrate to the cloud, an independent cloud-auditing service (run by the enterprise or, alternatively, a third party) monitors the enterprise’s cloud resources. This service regularly communicates bi-directionally with the gateway. Updates on enterprise data and workloads migrated to the cloud propagate from the enterprise to the auditing service, which communicates the results of its audits back to the enterprise, including, say, the health scores of various resources (such as data repositories and virtual machines).

Organization. Here, we describe several research projects that are components of this broad vision, starting with our design for an authenticated file system called Iris that allows migration of existing internal enterprise file systems into the cloud. Iris offers strong integrity and freshness guarantees of both file-system data and metadata accessed while users perform file-system operations. Iris minimizes the effects of network latency on file-system operations and is optimized for

typical file-system workloads (sequential file accesses). We then introduce our auditing framework, including Proofs of Retrievability (PoR) and related protocols that cryptographically verify the correctness of all cloud-stored data with minimal communication. (Remarkably, even against a cheating cloud, PoRs show that every bit of data stored in the cloud is intact.) We describe a dynamic PoR architecture that supports data updates in Iris and audit of physical-layer storage properties. We also show how to verify that cloud data is replicated across multiple hard drives with our Reversible Addition-Fragmentation chain Transfer, or RAFT, protocol. For further data protection we address the challenge of data availability in the face of cloud-service failures, including potentially malicious ones. We also describe our HAIL protocol, which distributes data redundantly across multiple cloud providers. HAIL is a cloud extension of the RAID principle, building reliable storage systems from inexpensive, unreliable components. We conclude with yet more challenges in securing cloud data.

Integrity Checking with Iris

Tenants commonly assume encrypting their data before sending it to the cloud is sufficient for securing it. Encryption provides strong confidential-

ity against a prying or breached cloud provider. It does not, however, protect against corruption of data due to software bugs or configuration errors, which require enforcement of a different property—data integrity—to ensure that data retrieved by a tenant is authentic, or has not been modified or corrupted by an unauthorized party. On its own, data integrity is relatively easy to achieve through cryptography (typically through Message-Authentication Codes, or MACs, on data blocks). But one critical yet subtle related security property—freshness—of data is often overlooked. Freshness ensures the latest updates are always propagated to the cloud and prevents rollback attacks in which stale versions of the data are presented to tenants.

Data freshness ensures retrieved data always reflects the most recent updates while preventing rollback attacks. Achieving data freshness is essential for protecting against misconfiguration errors or rollbacks that are caused intentionally and is the main technical challenge in building the Iris system we describe in the following section:

Iris design goals. Iris is an authenticated file system that supports migration of an enterprise-class distributed file system into the cloud, efficiently, transparently, and in a scalable manner. It is authenticated in the sense

that it enables an enterprise tenant to verify the integrity and freshness of retrieved data while performing file-system operations.

A key Iris design requirement is that it imposes on client applications no changes to file-system operations, including file read, write, update, and delete operations, as well as creation and removal of directories. That is, Iris does not require user machines to run modified applications. It also aims to achieve a slowdown in operation latency small enough to go unnoticed by users even when a large number of clients in the enterprise (on the order of hundreds and even thousands) issue file-system operations in parallel.

Iris architecture. An important challenge we faced when designing Iris is how to address the typically high network latency between an enterprise and the cloud. To reduce the effect of network latency on individual operation latency and the cost of network transfer to and from the cloud, Iris employs heavy caching on the enterprise side.

In Iris, a trusted gateway residing within the enterprise trust boundary, as in Figure 1, intermediates all communication from enterprise users to the cloud. The gateway caches data and meta-data blocks from the file system recently accessed by enterprise users. The gateway computes integrity checks, namely MACs, on data blocks. It also maintains integrity and freshness information for cached data consisting of parts of a tree-based authenticated data structure stored in the cloud.

The cloud maintains the distributed file system consisting of all enterprise files and directories. Iris is designed to use any existing back-end cloud-storage system transparently, without modification. The cloud also stores MACs for block-level integrity checks, as well as a tree-based cryptographic data structure needed to ensure the freshness of data blocks and the directory tree of the file system.

Integrity and freshness verification. To guarantee data integrity and freshness for an entire file system, Iris uses an authentication scheme consisting of two layers (see Figure 2). At the lower layer, it stores a MAC for each file block; file blocks are fixed-size

file segments of typical size 4KB. This structure enables random access to file blocks and verification of individual file-block integrity without accessing full files. For freshness, MACs are not sufficient, so Iris associates a counter or version number with each file block incremented on every block update¹⁵ and included in the block MAC. Different versions of a block can be distinguished through different version numbers. But for freshness, block version numbers must also be authenticated.

The upper layer of the authentication scheme is a Merkle tree tailored to the file system directory tree. The leaves of the Merkle tree store block-version numbers in compressed form. The authentication of data is separated from the authentication of block-version numbers to enable optimizations in the data structure. Internal nodes of the tree contain hashes of their children, as in a standard Merkle tree. The root of the Merkle tree must be maintained at all times within the enterprise-trust boundary at the gateway.

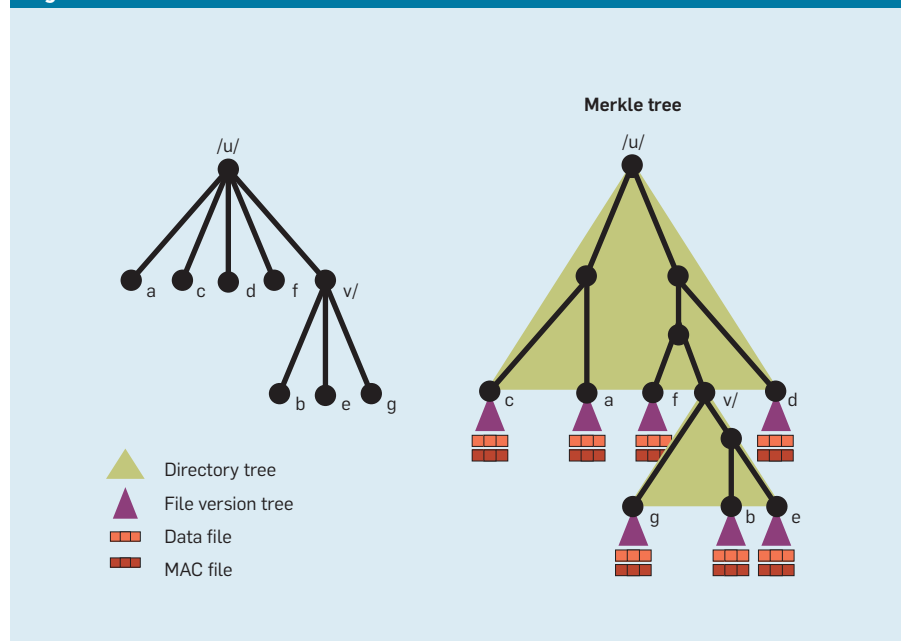
The tenant can efficiently verify the integrity and freshness of a file data block by checking the block MAC and the freshness of the block-version number. The tenant verifies the block-version number by accessing the sibling nodes on the path from the leaf storing the version number up to the root of the tree, recomputing all hash-

es on the path to the root and checking that the root matches the value stored locally. With a similar mechanism, the tenant can additionally verify the correctness of file paths in the file system and, more generally, of any other file system metadata (such as file names, number of files in a directory, and file-creation time).

This Merkle-tree-based structure has two distinctive features compared to other authenticated file systems: Support for existing file-system operations (Iris maintains a balanced binary tree over the file system directory structure to efficiently support existing file system calls); and support for concurrent operations (the Merkle tree supports efficient updates from multiple clients operating on the file system in parallel). Iris also optimizes for sequential file-block accesses; sequences of identical version counters are compressed into a single leaf.

The Iris authentication mechanism is practical and scalable; in a prototype system, a Merkle tree cache of only 10MB increases the system throughput by a factor of 3, so, compared to no caching, the throughput is fairly constant for approximately 100 clients executing operations on the file system in parallel; the operation-latency overhead introduced by processing at the gateway, including the integrity-checking mechanism, is at most 15%. These numbers are reported from a user-level

Figure 2. Authentication data structure in Iris.



implementation of Iris evaluated on commonly used benchmarks, including IOZone, sequential file reads and writes, and archiving of an entire directory structure.²⁰

Auditing Framework


Tools like Iris enable tenants to deploy their own security protections for data migrated to the cloud. But tenant self-protection is effective only to a point; for instance, even with Iris in place, a tenant's data is not safe against wholesale service-provider failure. Moreover, while Iris enables tenants to detect data loss resulting from a drive crash, it does not give them early warning of the probable precondition—a dangerous lack of provider storage redundancy.

A strong auditing framework is the cornerstone of tenant confidence in a service provider. Regulatory, reputational, and contractual assurances of provider safeguards are important, but ongoing technical assurances of solid security are irreplaceable.


Our research aims to develop a tenant-provider auditing relationship in which a tenant or external auditing service acting on the tenant's behalf, as in Figure 1, can continuously audit a provider to prove or disprove compliance with a given security policy. The provider responds to a challenge with a compact, real-time proof. Such auditing draws on the structure of a cryptographic challenge-response protocol; the tenant can rigorously verify the provider's response, obtaining a technically strong guarantee of policy compliance.

The challenge-response-style protocols described here cover a range of security properties, from data correctness to availability in the face of provider failures. We give a high-level overview of how they work and the guarantees they offer; we also briefly discuss their place in a larger vision, in which trusted hardware complements our auditing framework.

For concreteness, we mimic the cryptographic literature, referring to our canonical tenant as Alice and cloud provider as Bob. In our description here, Alice also acts as the auditor verifying properties of cloud-stored data, but in our more general framework, from Figure 1, the auditing pro-



When Alice (the tenant) stores data with Bob (the cloud), she wants to know that Bob has not let her data succumb to bit rot, storage-device failures, corruption by buggy software, or myriad other common threats to data integrity.



ocol could be executed by a separate entity—the auditing service.

Auditing data retrievability. When Alice (the tenant) stores data with Bob (the cloud), the most basic assurance she is likely to seek is that her data remains intact. She wants to know that Bob has not let her data succumb to bit rot, storage-device failure, corruption by buggy software, or myriad other common threats to data integrity. Because even a well-meaning Bob may be vulnerable to infection by malware, Alice also needs such assurance to be robust even if Bob cheats.

One strong cryptographic approach to assurance is the PoR,¹² a challenge-response protocol in which Bob proves to Alice that a given piece of data D stored in the cloud is intact and retrievable. While the Iris system enables verification of data integrity for data retrieved from the cloud in the course of performing regular file-system operations, a PoR enables verification of an entire data collection without first retrieving it from the cloud.

This goal might seem counterintuitive at first, even impossible. A PoR can demonstrate with a compact proof (on the order of, say, hundreds of bytes) that every single bit of data is intact and accessible to Alice, even if it is very large (gigabytes or more).

Building a PoR, step by step. Here, to give a sense of how a PoR works, we develop a construction in stages. An obvious candidate approach is for Alice to simply store a cryptographic hash $c = h(D)$ of data D in the cloud. To verify that D is intact, she challenges Bob to send her c . However, this idea involves two problems: Bob can cheat, storing c and throwing away D , though a refinement incorporating a secret “nonce” into the hash would address it. Efficiency considerations are a more fundamental drawback; to generate c from D authentically, Bob must hash all of D , a resource-intensive process if D is big.

An alternative approach is for Alice to sample data blocks and verify their correctness, in effect spot-checking her data. Now let r_i denote the i^{th} data block; blocks are fixed-size segments of data with typical size 4KB. Before storing the data into the cloud, Alice locally retains a randomly selected block r_i . To challenge Bob, she sends him the block-index i and asks him to

transmit r_i , which she verifies against her local copy. To amplify the probability of detecting data corruption, Alice can request multiple blocks with independent random indices i_1, i_2, \dots, i_m simultaneously.

Bob now has to touch only a small portion of the data to respond to a challenge, solving the resource-consumption problem with hashing. If D , or a large chunk of D , is missing or corrupted, Alice will detect this fact with high probability, as desired. However, the scheme still involves two drawbacks:

First, while Alice can detect large corruptions with high probability, she is unlikely to detect small corruptions of, say, limited bit rot, even with multiple challenge blocks. Suppose her data has one million blocks and she challenges Bob on 10 randomly selected blocks; the probability of Alice detecting a one-bit error would be less than 0.001%. And second, she must use fresh blocks for each challenge, so Bob cannot predict future challenges from past ones. So, if Alice plans to challenge Bob many times, she must store a considerable amount of data locally.

To solve the first, we can appeal to an error-correcting code, a technique for adding redundancy to some piece of data D (called “message”), yielding encoded data D^* (called “code word”). D^* is often constructed by appending what are called “parity blocks” to the end of D . If a limited portion of the encoded data D^* is corrupted or erased, a decoding function can still be applied to restore the original data D . The expansion ratio ($|D^*| = |D|$) and amount of tolerable corruption depend on the code parameters. For the sake of the example, though, consider an error-correcting code that expands data by 10%— $|D^*| = |D| = 1.1$ —and can successfully decode it provided that at most 10% of the blocks in D^* are corrupted.

Now, if Alice stores D^* instead of D , she is assured her data will be lost only if a significant fraction of her data is corrupted or erased. That is, for a single bit of D to be irretrievably corrupted, a large chunk of D^* must be corrupted. Use of error-correcting effectively amplifies the power of Alice’s challenges. Suppose, for instance, she uses a code that corrects up to 10% corruption. Now, issuing 10 challenges against a one-million-block data col-

lection will result in detection of any irretrievably corrupted bits in D with probability over 65%—a vast improvement over 0.001%.

Solving the problem of excessive local storage is fairly easy. Using a locally stored secret key k , Alice can compute MACs—secret-key digital signatures—over data blocks r_1, r_2, \dots, r_n . She can safely have Bob store MACs c_1, c_2, \dots, c_n alongside D^* . To verify the correctness of a block r_i , Alice uses k and c_i . So Alice needs to store only the key k . Figure 3 shows the data stored by Alice and Bob in this PoR construction.

Concerning other challenges in constructing a practical PoR, Alice can send, say, a key to a pseudorandom function during a challenge, based on which Bob can infer the position of all challenged blocks. Bob can also aggregate multiple response blocks into one, rendering transmission and verification more efficient. Additionally, making error-correcting practical for large data collections requires coding and cryptographic tricks, though the solution presented here provides much of the intuition behind a full-blown PoR construction.

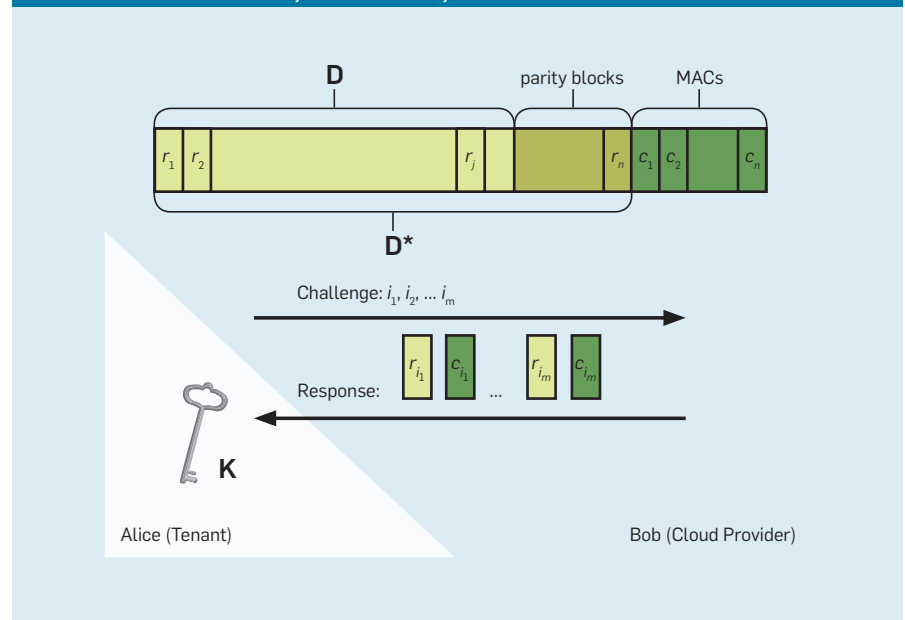
A PoR can be used to verify the integrity and correctness of any type of data collection stored in the cloud, including file systems and key-value stores. Its salient property is it gives Alice strong guarantees about the correct-

ness of an entire data collection using minimal computation and bandwidth for auditing. The auditing protocol could be performed by a third-party auditing service.

Variants. Several PoR variants are available for auditing data integrity: for example, a Proof of Data Possession (PDP)² enables public verification of data integrity by employing public-key cryptography. Compared with PoRs, PDP provides detection of only large amounts of data corruption, without a recovering mechanism. PDP and PoR protocols can be privately or publicly verifiable. In a privately verifiable protocol, auditing can be performed only by the party who knows the secret key used for encoding the data. In contrast, in a publicly verifiable protocol, auditing can be performed by any third-party service, at the cost of more computationally expensive encoding and auditing protocols. The term Proof of Storage (PoS)³ has evolved as a convenient catchall for PoRs and PDPs. Most PoS protocols can be combined with other cryptographic protections (such as encryption for data confidentiality) to achieve a suite of cloud-data protections.¹³

PoRs in Iris. A basic PoR, as described earlier, has a notable limitation: an inability to handle data updates gracefully. Changing a single data block in D requires propagation

Figure 3. Data stored by Alice and Bob in a PoR. To issue a challenge to Bob, Alice indicates positions i_1, \dots, i_m to Bob. Bob returns data blocks r_{i_1}, \dots, r_{i_m} , along with MACs c_{i_1}, \dots, c_{i_m} . Alice verifies the correctness of r_{ij} against MAC c_{ij} , for all $j \in \{1, \dots, m\}$ using secret key k .



of changes across the parity blocks of D^* . So a basic PoR is efficient only for checks on static data (such as archived data). The situation is somewhat better without error correction; researchers have proposed (asymptotically efficient) PDP systems that support data updates.⁸ But support for updates also rules out recovering from a small amount of data corruption.

It is natural, then, to ask whether we can have the best of both worlds—a PoR for dynamically changing data. Yes is the answer.

Check values (MACs or digital signatures) pose the first major challenge in supporting a dynamic PoR/PDP. Not only must they be updated in conjunction with data-block updates, but, when verifying them, Alice must be able to determine they are both correct and fresh. The Iris system is designed precisely to tackle the tricky problem of efficient freshness verification for file systems, making it ideal for building a dynamic PoR.

An even more formidable challenge is updating error-correcting blocks as data blocks change. That is, to protect against targeted corruption by Bob, the structure of the error-correcting code, and thus the pattern of parity block updates, must be hidden from him. Encryption is no help; basic encryption hides data, not access patterns.

The way out of this conundrum is a model shift inspired by the deployment objectives of Iris. While PoR designers generally aim to keep Alice’s storage to a minimum, Iris aims at enterprise-

class cloud deployment. When Alice is a company, rather than an individual, substantial tenant-side resources are a reasonable expectation.

The key idea for dynamic PoR layered on Iris is to have Alice cache parity blocks locally, on the enterprise side, and periodically back them up to the cloud. This approach conceals individual parity-block updates from Bob, as well as the code structure. It has another advantage, too: Alice’s updates to parity blocks can be made locally. As a single data-block update results in multiple parity-block updates, the ability to make updates locally greatly reduces communication between Alice and Bob.

The result is an enhancement such that Iris not only verifies file integrity and freshness but can also check (efficiently) whether an entire file system is intact, down to the last bit. In addition, if corruptions to data are found (through auditing or through integrity verification using the Merkle tree structure described earlier), Iris can recover corrupted blocks from the additional redundancy provided by the erasure code. Iris provides strong guarantees of detection and remediation of data corruption, resulting in retrievability of an entire file system stored in the cloud. A great Iris benefit is that its parameters for the erasure code and the communication during an audit can be adjusted for a desired level of recoverability.

While construction in Iris provides the first practical solution to a dynamic PoR protocol, it relies on some amount

of local storage maintained by the tenant; in our Iris instantiation the client maintains $O(vn)$ local storage for a file system with n data blocks. The problem of constructing a dynamic PoR protocol with constant storage at the client-side is the major remaining theoretical cryptographic research challenge in auditing data retrievability.

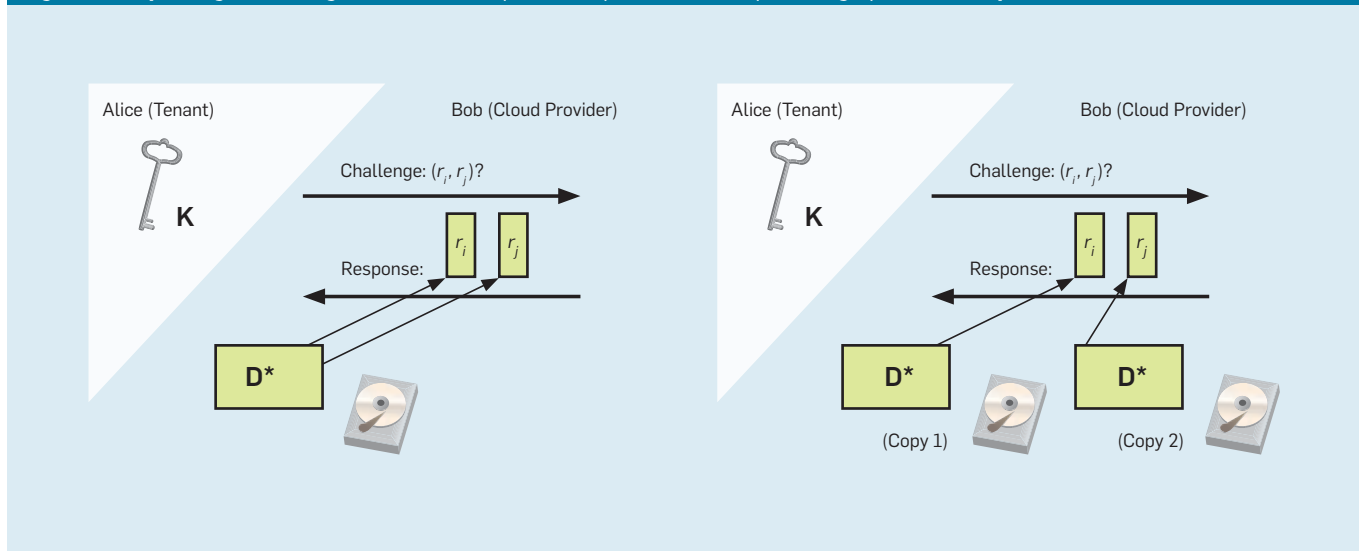
Auditing of drive-failure resilience.

Detecting data loss via Proofs of Storage (PoS) is helpful, though prevention is best. One major cause of data loss is drive crashes. In a large data center with hundreds of thousands of drives, drive failure is the norm rather than the exception. With 2%–3% annual failure rates published by manufacturers and even higher numbers observed in the field,¹⁹ a large data center is likely to experience thousands of drive failures each year.

Services (such as Amazon S3) claim to store files in triplicate. But how can such a claim be verified remotely? At first glance, it seems impossible. Suppose Alice wants to verify that Bob is storing three copies of her data. Downloading the three copies would not work; if Bob is cheating and storing only one copy of the data, he can simply transmit that one copy three times. There is, however, a very simple solution: Alice can encrypt each copy of her data under a separate key, yielding three distinct encryptions. Executing a PoS against each encrypted version ensures the existence of three distinct copies.

In many cases, though, Alice does not want to have to store her data in en-

Figure 4. Responding to challenges from one disk (on the left) and two disks (on the right) in the RAFT protocol.



encrypted form. She may want Bob to be able to process her data for her or make it available to her friends. More important, the existence of three distinct copies does not per se ensure resilience to drive crashes. All three copies could be sitting on the same drive, after all. So how can Alice verify that there are three distinct copies of the data, each on a different drive?

A striking feature of the problem is that it is primarily physical, not logical. The objective is not to verify the encoding or mere existence of data but its disposition on a physical substrate.

RAFT⁷ is the solution, allowing Alice to verify that Bob has stored some piece of data D so it can survive up to t drive failures, for a desired parameter t . RAFT allows D to be dispersed using erasure coding, a more space-efficient technique than maintaining full file copies. RAFT operates specifically on data stored in rotational drives, exploiting their performance limitations as a bounding parameter. The more drives across which Bob has striped D , the faster he can respond to a challenge. In particular, RAFT makes use of bounds on the seek time of a rotational drive. Alice transforms her data D using an erasure code into encoded data D^* . D^* can be striped across c drives such that if any t fail, D can be recovered. She asks Bob to store D^* across c drives.

To verify resilience to t drive failures, Alice challenges Bob to fetch a set of n randomly distributed blocks from D^* . Suppose Bob stores D^* on d drives.

Each block fetch incurs a seek (assuming the random blocks are spread apart at a large distance). So on average, if a seek takes time μ , Bob's total fetch time is $\mu n/d$. If $d < c$, then his response time will take $\mu n(1/d-1/c)$ longer than expected, on average. By measuring Bob's response time, then, Alice can determine whether he is indeed using c drives, as required (see Figure 4).

While we do not go into detail here, many complications arise in real-world settings. Variations in drive performance, as well as in network latency, between Alice and Bob must be measured carefully. Sensitive statistical analysis and structuring of challenges is required to accommodate these variations.

Hardware roots of trust. Another approach to assurance within the challenge-response framework we explore here is a hardware root of trust, as supported by, say, Trusted Platform Modules that permit a tenant to verify remotely, via a challenge-response protocol, a provider is executing a particular software stack.

But hardware roots of trust cannot directly enforce guarantees on storage integrity or reliability. Even if Bob's servers are configured precisely, as specified by Alice, and even if Alice controls their operation, she obtains no direct guarantee as to their corresponding storage subsystem. For instance, the only way for Alice to determine a file F is intact in storage without full-file inspection is to perform a PoR. The same holds for properties verified by Iris, HAIL, and RAFT, none guaranteed

solely through trustworthy execution environments.

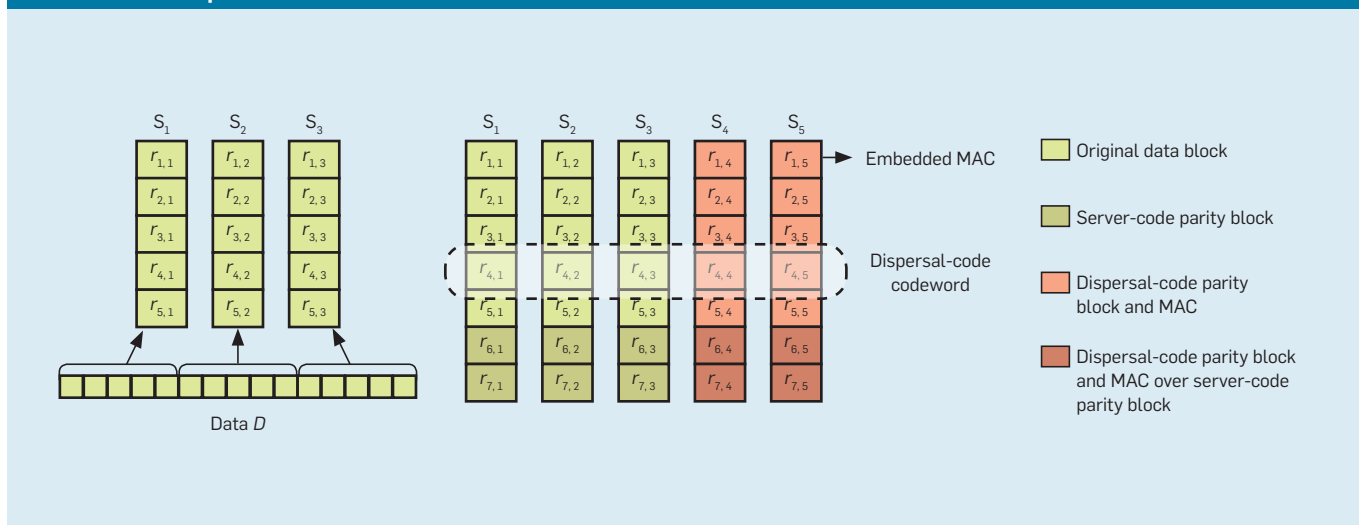
Enhancing Data Availability with HAIL

We have described an auditing framework that offers tenants visibility into the operations of the cloud and verification of some properties of their cloud-side data. But what happens if the cloud provider fails to respond correctly to an audit due to data loss? A major impediment to cloud adoption by enterprises is temporary lack of availability by the provider or even permanent failure. This is a real threat, as illustrated by catastrophic provider failures resulting in massive customer data loss.²³

We designed HAIL⁶ specifically to address this challenge, predicated on the idea that it is wise to distribute data across multiple cloud providers for continuous availability. HAIL thus leverages multiple cloud providers to construct a reliable, cost-effective cloud-storage service from unreliable components. The idea is similar in flavor to RAID,¹⁶ which creates reliable storage arrays from unreliable hard drives. HAIL extends the idea into the cloud, the main differences being its support for a stronger adversarial model and a higher-level abstraction.

HAIL works by promptly detecting and recovering from data corruption. The tenant (or third-party service) periodically audits individual cloud providers toward this end. HAIL auditing is lightweight in terms of both bandwidth

Figure 5. Encoding of data D : on the left, original data is represented as a matrix; on the right, encoded data with parity blocks is added for both server and dispersal codes.



and computation. Using the redundancy embedded across different cloud providers, the tenant (or third party) remediates corruption detected in a subset of providers. HAIL is reactive, rather than proactive, meaning it remediates data only on detected corruption.

System model. In HAIL a tenant like Alice distributes her data with embedded redundancy to a set of n cloud providers: S_1, \dots, S_n . In our model, data generated by all enterprise users is transmitted to the gateway, as in Figure 1. The gateway performs some data encoding, described in the following paragraphs, optionally encrypts data, and distributes a data fragment to each cloud provider. HAIL operates through an adversarial model in which a strong mobile adversary can corrupt all cloud providers over time. But within a single epoch (a predetermined period of fixed length) the adversary can corrupt at most b out of n servers, for some $b < n$.

HAIL encoding. Figure 5 outlines the encoding of data in HAIL. To provide resilience against cloud-provider failure, the gateway splits the data into fixed-size blocks and encodes it with a new erasure code we call “dispersal code.” The figure includes a matrix representation of the data on the left, resulting in an encoded matrix on the right. Each row in the encoded matrix is a stripe or code word obtained by applying the dispersal code. Each row contains the original data blocks, as well as new parity blocks obtained with the dispersal code. Each matrix column is stored at a different cloud provider. The dispersal code guarantees the original data can be reconstructed, given up to b cloud provider failures (and $n - b$ intact columns).

However, a single layer of encoding is not sufficient to guarantee data availability and integrity in HAIL’s strong adversarial model. Consider this attack: The adversary corrupts b new servers in each epoch, picks a particular row index i , and corrupts the corresponding block r_{ij} at server S_j . After $\lceil n/b \rceil$ epochs, the adversary corrupts all servers and the entire row i in the encoded matrix from Figure 5. In this case, the redundancy of the dispersal code is not helpful in recovering the corrupted row, and the entire data D is permanently corrupted.



HAIL is an extension of RAID into the cloud, distributing data across multiple cloud providers to achieve continuous availability.



How can the system prevent such a creeping-corruption attack? By simply auditing a few randomly selected blocks at each server, the probability that the tenant would discover the corruption of blocks in a single row of the encoded matrix is quite low. Therefore, another encoding layer we call a “server code” is needed within each server. The server code adds additional redundancy (parity blocks) to each column in the encoded matrix representation. The role of the server code is to recover from a small amount of corruption at each cloud provider that would otherwise be undetectable through the auditing protocol.

To prevent adversarial data corruption, the tenant also needs to store MACs on data blocks in the cloud. With a new technique we call an “integrity-protected dispersal code,” parity blocks of the dispersal code can themselves be used as MACs on the rows, thus reducing the storage overhead for integrity protection.

Auditing and recovering from failures. In HAIL, the gateway (or an external auditing service, as in Figure 1) periodically audits the correctness of the cloud data by contacting all cloud providers. The gateway sends a random row index i as a challenge to each cloud provider, and verifies, upon receiving the responses r_{ij} , for j in $\{1, \dots, n\}$ the correctness of the entire row. It is also possible to aggregate multiple responses (multiple randomly selected blocks) from each server to reduce bandwidth and amplify the probability of failure detection.

When data corruption at one or more cloud providers is detected, the corrupted data can be reconstructed at the tenant side using the two encoding layers—the dispersal and server code. Data reconstruction is an expensive protocol, one rarely invoked (only upon detection of data corruption).

With its encoding, auditing, and recovery mechanisms, HAIL provides resilience against a strong mobile adversary that might potentially corrupt all providers over time. However, a limitation of HAIL is that, as designed, it does not handle file updates gracefully. Rather, it is most suited to archival data, or data stored in the cloud for retention purposes and not regularly modified. More-efficient versions of

HAIL can be constructed under a weaker adversarial model that may be practical for short-term data storage.

Conclusion

We have described new techniques that secure cloud data by ensuring a range of protections, from integrity and freshness verification to high data availability. We also proposed an auditing framework that offers tenants visibility into the correct operation of the cloud. These techniques enable an extension of the trust perimeter from enterprise internal data centers into public clouds, as in Figure 1. Our hope is these techniques will alleviate some of the concern over security in the cloud and facilitate migration of enterprise resources into public clouds. We conclude here by mentioning several open problems of interest in this context:

Performing computations over tenants' encrypted data. We have emphasized data integrity and availability, but data confidentiality is a major open problem. General computation over a tenant's encrypted data is possible using a technique we call "fully homomorphic encryption," though this breakthrough¹⁰ is not yet practical. Weaker, custom techniques can achieve specific functionalities (such as searches¹⁴ and general SQL queries¹⁷) over encrypted data.

The impossibility of general computations over multiple tenants' encrypted data using only cryptographic techniques and no interaction among tenants was shown in van Dijk and Juels.²² A promising area of research is the design of custom protocols for applications involving multiple tenants' data (such as data mining over multiple institutions' medical records or financial transactions). Combining secure hardware architectures with cryptography (such as secure multi-party computation protocols) offers significant potential.

Ensuring tenant isolation. Cloud co-tenancy with attackers can jeopardize tenant data, as shown in Ristenpart et al.,¹⁸ which explored cache-based side channels for data exfiltration. The risks of co-tenancy in storage systems (such as storage-system side channels) is an unexplored vector of attack now deserving investigation, in our view.

One approach to co-tenancy risks is to isolate tenants by implementing virtual private cloud (VPC) abstractions within a public cloud. HomeAlone²⁴ enables tenants to verify remotely that a VPC is strongly enforced at the host level, in the sense of creating physical isolation of a tenant's workloads. While physical isolation offers a solution for extremely sensitive workloads, it can undercut the financial benefits of tenant sharing of computing resources. For this reason, solutions offering tenant isolation and enabling the sharing of cloud resources at the same time are extremely important. Trusted hardware may also play an important role, along with tight enforcement of logical isolation abstractions throughout the software stack, including hypervisor and OS, and across the cloud fabric.

Geolocation of data. Of particular commercial interest is the open problem of remote verification of the geographical location of cloud data. The motivation is regulatory compliance, with many laws requiring providers keep customer data within, say, national boundaries.⁹ Given the challenge of ensuring that data is not duplicated, any solution probably requires a trusted data-management system via, say, trusted hardware, and localizing the pieces of the system is an interesting challenge. Geolocation of trusted hardware via remote timing from trusted anchor points seems a key avenue of exploration. □

References

1. Armbrust, M., Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., and Zaharia, M. A view of cloud computing. *Commun. ACM* 53, 4 (Apr. 2010), 50–58.
2. Ateniese, G., Burns, R., Curtmola, R., Herring, J., Kissner, L., Peterson, Z., and Song, D. Provable data possession at untrusted stores. In *Proceedings of the 14th ACM Conference on Computer and Communications Security* (Alexandria, VA, Oct. 28–31). ACM Press, New York, 2007, 598–609.
3. Ateniese, G., Kamara, S., and Katz, J. Proofs of storage from homomorphic identification protocols. In *Proceedings of the Conference on Advances in Cryptology Lecture Notes in Computer Science 5912* (Tokyo, Dec. 6–10). Springer, 2009, 319–333.
4. Baker, M., Shah, M., Rosenthal, D.S.H., Roussopoulos, M., Maniatis, P., Giuli, T., and Bungate, P. A fresh look at the reliability of long-term digital storage. In *Proceedings of the European Conference on Computer Systems* (Leuven, Belgium, Apr. 18–21). ACM Press, New York, 2006, 221–234.
5. Blumenthal, M. Is security lost in the cloud? *Communications and Strategies* 1, 81 (2011), 69–86.
6. Bowers, K.D., Juels, A., and Oprea, A. HAIL: A high-availability and integrity layer for cloud storage. In *Proceedings of the 16th ACM Conference on Computer and Communications Security* (Chicago, Nov. 9–13). ACM Press, New York, 2009, 187–198.
7. Bowers, K.D., van Dijk, M., Juels, A., Oprea, A., and Rivest, R.L. How to tell if your cloud files are vulnerable to drive crashes. In *Proceedings of the 18th*

- ACM Conference on Computer and Communications Security (Chicago, Oct. 17–21). ACM Press, New York, 2011, 501–514.
8. Erway, C., Kupcu, A., Papamanthou, C., and Tamassia, R. Dynamic provable data possession. In *Proceedings of the 16th ACM Conference on Computer and Communications Security* (Chicago, Nov. 9–13). ACM Press, New York, 2009, 213–222.
9. European Parliament. *Directive 95/46/EC of the European Parliament of the Council* (Data Protection Directive), 1995; <http://bit.ly/5eLDdi>
10. Gentry, C. Computing arbitrary functions of encrypted data. *Commun. ACM* 53, 3 (Mar. 2010), 97–105.
11. Helft, M. Google confirms problems with reaching its services. *The New York Times* (May 14, 2009); <http://www.developmentguruji.com/news/99/Google-confirms-problems-with-reaching-its-services.html>
12. Juels, A. and Kaliski, B. PORs: Proofs of retrievability for large files. In *Proceedings of the 14th ACM Conference on Computer and Communications Security* (Alexandria, VA, Oct. 28–31). ACM Press, New York, 2007, 584–597.
13. Kamara, S. and Lauter, K. Cryptographic cloud storage. In *Proceedings of Financial Cryptography: Workshop on Real-Life Cryptographic Protocols and Standardization, Lecture Notes in Computer Science 6054* (Tenerife, Canary Islands, Spain, Jan. 25–28). Springer, 2010, 136–149.
14. Kamara, S., Papamanthou, C., and Roeder, T. Cs2: A Searchable Cryptographic Cloud Storage System. Technical Report MSR-TR-2011-58. Microsoft, Redmond, WA, 2011.
15. Oprea, A. and Reiter, M.K. Integrity checking in cryptographic file systems with constant trusted storage. In *Proceedings of the 16th Usenix Security Symposium* (Boston, Aug. 6–10). USENIX Association, Berkeley, CA, 2007, 183–198.
16. Patterson, D., Gibson, G., and Katz, R. A case for redundant arrays of inexpensive disks (RAID). *SIGMOD Record* 17, 3 (Sept. 1988), 109–116.
17. Popa, R.A., Redfield, C.M.S., Zeldovich, N., and Bakrishnan, H. CryptDB: Protecting confidentiality with encrypted query processing. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles* (Cascais, Portugal, Oct. 23–26). ACM Press, New York, 2011, 85–100.
18. Ristenpart, T., Tromer, E., Shacham, H., and Savage, S. Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds. In *Proceedings of the 16th ACM Conference on Computer and Communications Security* (Chicago, Nov. 9–13). ACM Press, New York, 2009, 199–212.
19. Schroeder, B. and Gibson, G. Disk failures in the real world: What does an MTTF of 1,000,000 hours mean to you? In *Proceedings of the Fifth USENIX Conference on File and Storage Technologies* (San Jose, CA, Feb. 13–16). USENIX Association, Berkeley, CA, 2007, 1–16.
20. Stefanov, E., van Dijk, M., Oprea, A., and Juels, A. Iris: A scalable cloud file system with efficient integrity checks. In *Proceedings of the 28th Annual Computer Security Applications Conference* (Orlando, FL, Dec. 3–7, 2012).
21. Stern, A. *Update from Amazon regarding Friday S3 downtime*. CenterNetworks, Feb. 16, 2008; <http://www.centrerworks.com/amazon-s3-downtime-update>
22. van Dijk, M. and Juels, A. On the impossibility of cryptography alone for privacy-preserving cloud computing. In *Proceedings of the HOTSEC Workshop on Hot Topics in Security* (Washington, D.C., Aug. 11–13). USENIX Association, Berkeley, CA, 2010.
23. Wingfield, N. Microsoft, T-Mobile stumble with Sidekick glitch. *The Wall Street Journal* (Oct. 11, 2009); <http://online.wsj.com/article/SB10001424052748703790404574467431941990194.html>
24. Zhang, Y., Juels, A., Oprea, A., and Reiter, M.K. HomeAlone: Co-residency detection in the cloud via side-channel analysis. In *Proceedings of the IEEE Symposium on Security and Privacy* (Berkeley, CA, May 22–25). IEEE Computer Society Press, 2011, 313–328.

Ari Juels (ajuels@rsa.com) is Chief Scientist of RSA and Director of RSA Laboratories, Cambridge, MA.

Atina Oprea (aoprea@rsa.com) is a research scientist in RSA Laboratories, Cambridge, MA.

DOI:10.1145/2408776.2408794

Software techniques that tolerate latency variability are vital to building responsive large-scale Web services.

BY JEFFREY DEAN AND LUIZ ANDRÉ BARROSO

The Tail at Scale

SYSTEMS THAT RESPOND to user actions quickly (within 100ms) feel more fluid and natural to users than those that take longer.³ Improvements in Internet connectivity and the rise of warehouse-scale computing systems² have enabled Web services that provide fluid responsiveness while consulting multi-terabyte datasets spanning thousands of servers; for example, the Google search system updates query results interactively as the user types, predicting the most likely query based on the prefix typed so far, performing the search and showing the results within a few tens of milliseconds. Emerging augmented-reality devices (such as the Google Glass prototype⁷) will need associated Web services with even greater responsiveness in order to guarantee seamless interactivity.

It is challenging for service providers to keep the tail of latency distribution short for interactive services as the size and complexity of the system scales up or

as overall use increases. Temporary high-latency episodes (unimportant in moderate-size systems) may come to dominate overall service performance at large scale. Just as fault-tolerant computing aims to create a reliable whole out of less-reliable parts, large online services need to create a predictably responsive whole out of less-predictable parts; we refer to such systems as “latency tail-tolerant,” or simply “tail-tolerant.” Here, we outline some common causes for high-latency episodes in large online services and describe techniques that reduce their severity or mitigate their effect on whole-system performance. In many cases, tail-tolerant techniques can take advantage of resources already deployed to achieve fault-tolerance, resulting in low additional overhead. We explore how these techniques allow system utilization to be driven higher without lengthening the latency tail, thus avoiding wasteful overprovisioning.

Why Variability Exists?

Variability of response time that leads to high tail latency in individual components of a service can arise for many reasons, including:

Shared resources. Machines might be shared by different applications contending for shared resources (such as CPU cores, processor caches, memory bandwidth, and network bandwidth), and within the same application different requests might contend for resources;

Daemons. Background daemons may use only limited resources on average but when scheduled can generate multi-millisecond hiccups;

» key insights

- Even rare performance hiccups affect a significant fraction of all requests in large-scale distributed systems.
- Eliminating all sources of latency variability in large-scale systems is impractical, especially in shared environments.
- Using an approach analogous to fault-tolerant computing, tail-tolerant software techniques form a predictable whole out of less-predictable parts.



Global resource sharing. Applications running on different machines might contend for global resources (such as network switches and shared file systems);

Maintenance activities. Background activities (such as data reconstruction in distributed file systems, periodic log compactions in storage systems like BigTable,⁴ and periodic garbage collection in garbage-collected languages) can cause periodic spikes in latency; and

Queueing. Multiple layers of queueing in intermediate servers and network switches amplify this variability.

Increased variability is also due to several hardware trends:

Power limits. Modern CPUs are designed to temporarily run above their average power envelope, mitigating thermal effects by throttling if this activity is sustained for a long period;⁵

Garbage collection. Solid-state storage devices provide very fast random read access, but the need to periodically garbage collect a large number of data blocks can increase read latency by a factor of 100 with even a modest level of write activity; and

Energy management. Power-saving modes in many types of devices save considerable energy but add additional latency when moving from inactive to active modes.

Component-Level Variability Amplified By Scale

A common technique for reducing latency in large-scale online services is to parallelize sub-operations across many different machines, where each sub-operation is co-located with its portion of a large dataset. Parallelization happens by fanning out a request from a root to a large number of leaf servers and merging responses via a request-distribution tree. These sub-operations must all complete within a strict deadline for the service to feel responsive.

Variability in the latency distribution of individual components is magnified at the service level; for example, consider a system where each server typically responds in 10ms but with a 99th-percentile latency of one second. If a user request is handled on just one such server, one user request in 100 will be slow (one second). The figure here outlines how service-level latency in this hypothetical scenario is affected by very

modest fractions of latency outliers. If a user request must collect responses from 100 such servers in parallel, then 63% of user requests will take more than one second (marked “x” in the figure). Even for services with only one in 10,000 requests experiencing more than one-second latencies at the single-server level, a service with 2,000 such servers will see almost one in five user requests taking more than one second (marked “o” in the figure).

Table 1 lists measurements from a real Google service that is logically similar to this idealized scenario; root servers distribute a request through intermediate servers to a very large number of leaf servers. The table shows the effect of large fan-out on latency distributions. The 99th-percentile latency for a single random request to finish, measured at the root, is 10ms. However, the 99th-percentile latency for all requests to finish is 140ms, and the 99th-percentile latency for 95% of the requests finishing is 70ms, meaning that waiting for the slowest 5% of the requests to complete is responsible for half of the total 99%-percentile latency. Techniques that concentrate on these slow outliers can yield dramatic reductions in overall service performance.

Overprovisioning of resources, careful real-time engineering of software, and improved reliability can all be used at all levels and in all components to reduce the base causes of variability. We next describe general approaches useful for reducing variability in service responsiveness.

Reducing Component Variability

Interactive response-time variability can be reduced by ensuring interactive requests are serviced in a timely manner through many small engineering decisions, including:

Differentiating service classes and higher-level queueing. Differentiated service classes can be used to prefer scheduling requests for which a user is waiting over non-interactive requests. Keep low-level queues short so higher-level policies take effect more quickly; for example, the storage servers in Google’s cluster-level file-system software keep few operations outstanding in the operating system’s disk queue, instead maintaining their own priority queues of pending disk requests. This shallow

queue allows the servers to issue incoming high-priority interactive requests before older requests for latency-insensitive batch operations are served.

Reducing head-of-line blocking. High-level services can handle requests with widely varying intrinsic costs. It is sometimes useful for the system to break long-running requests into a sequence of smaller requests to allow interleaving of the execution of other short-running requests; for example, Google’s Web search system uses such time-slicing to prevent a small number of very computationally expensive queries from adding substantial latency to a large number of concurrent cheaper queries.

Managing background activities and synchronized disruption. Background tasks can create significant CPU, disk, or network load; examples are log compaction in log-oriented storage systems and garbage-collector activity in garbage-collected languages. A combination of throttling, breaking down heavyweight operations into smaller operations, and triggering such operations at times of lower overall load is often able to reduce the effect of background activities on interactive request latency. For large fan-out services, it is sometimes useful for the system to synchronize the background activity across many different machines. This synchronization enforces a brief burst of activity on each machine simultaneously, slowing only those interactive requests being handled during the brief period of background activity. In contrast, without synchronization, a few machines are always doing some background activity, pushing out the latency tail on all requests.

Missing in this discussion so far is any reference to caching. While effective caching layers can be useful, even a necessity in some systems, they do not directly address tail latency, aside from configurations where it is guaranteed that the entire working set of an application can reside in a cache.

Living with Latency Variability

The careful engineering techniques in the preceding section are essential for building high-performance interactive services, but the scale and complexity of modern Web services make it infeasible to eliminate all latency variability. Even if such perfect behavior could be achieved in isolated environments,

systems with shared computational resources exhibit performance fluctuations beyond the control of application developers. Google has therefore found it advantageous to develop tail-tolerant techniques that mask or work around temporary latency pathologies, instead of trying to eliminate them altogether. We separate these techniques into two main classes: The first corresponds to within-request immediate-response techniques that operate at a time scale of tens of milliseconds, before longer-term techniques have a chance to react. The second consists of cross-request long-term adaptations that perform on a time scale of tens of seconds to minutes and are meant to mask the effect of longer-term phenomena.

Within Request Short-Term Adaptations

A broad class of Web services deploy multiple replicas of data items to provide additional throughput capacity and maintain availability in the presence of failures. This approach is particularly effective when most requests operate on largely read-only, loosely consistent datasets; an example is a spelling-correction service that has its model updated once a day while handling thousands of correction requests per second. Similarly, distributed file systems may have multiple replicas of a given data chunk that can all be used to service read requests. The techniques here show how replication can also be used to reduce latency variability within a single high-level request:

Hedged requests. A simple way to curb latency variability is to issue the same request to multiple replicas and use the results from whichever replica responds first. We term such requests “hedged requests” because a client first sends one request to the replica believed to be the most appropriate, but then falls back on sending a secondary request after some brief delay. The client cancels remaining outstanding requests once the first result is received. Although naive implementations of this technique typically add unacceptable additional load, many variations exist that give most of the latency-reduction effects while increasing load only modestly.

One such approach is to defer sending a secondary request until the first

request has been outstanding for more than the 95th-percentile expected latency for this class of requests. This approach limits the additional load to approximately 5% while substantially shortening the latency tail. The technique works because the source of latency is often not inherent in the particular request but rather due to other forms of interference. For example, in a Google benchmark that reads the values for 1,000 keys stored in a BigTable table distributed across 100 different servers, sending a hedging request after a 10ms delay reduces the 99.9th-percentile latency for retrieving all 1,000 values from 1,800ms to 74ms while sending just 2% more requests. The overhead of hedged requests can be further reduced by tagging them as lower priority than the primary requests.

Tied requests. The hedged-requests technique also has a window of vulner-

ability in which multiple servers can execute the same request unnecessarily. That extra work can be capped by waiting for the 95th-percentile expected latency before issuing the hedged request, but this approach limits the benefits to only a small fraction of requests. Permitting more aggressive use of hedged requests with moderate resource consumption requires faster cancellation of requests.

A common source of variability is queueing delays on the server before a request begins execution. For many services, once a request is actually scheduled and begins execution, the variability of its completion time goes down substantially. Mitzenmacher¹⁰ said allowing a client to choose between two servers based on queue lengths at enqueue time exponentially improves load-balancing performance over a uniform random scheme. We advocate not

Probability of one-second service-level response time as the system scales and frequency of server-level high-latency outliers varies.

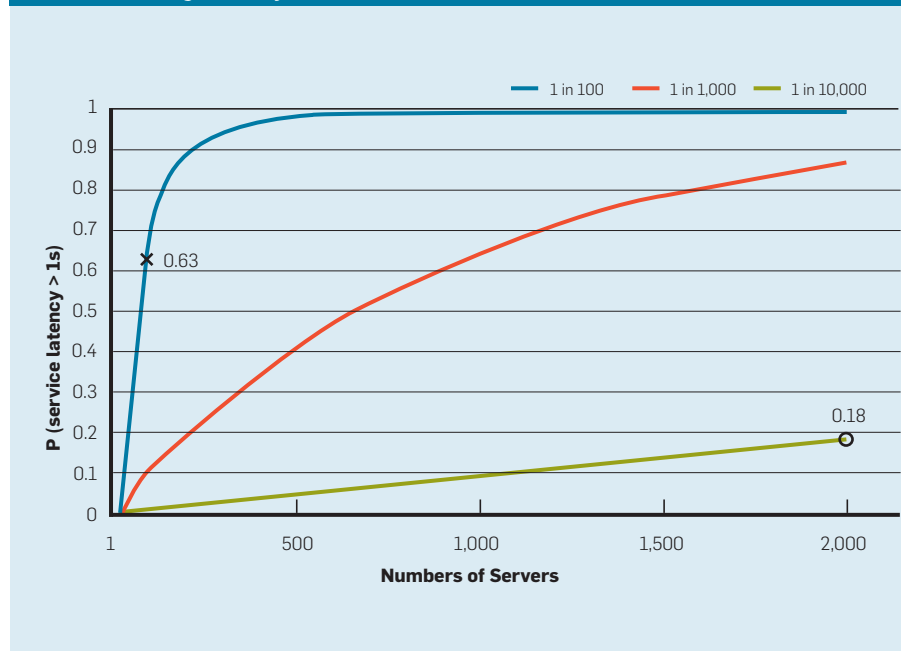


Table 1. Individual-leaf-request finishing times for a large fan-out service tree (measured from root node of the tree).

	50%ile latency	95%ile latency	99%ile latency
One random leaf finishes	1ms	5ms	10ms
95% of all leaf requests finish	12ms	32ms	70ms
100% of all leaf requests finish	40ms	87ms	140ms

choosing but rather enqueueing copies of a request in multiple servers simultaneously and allowing the servers to communicate updates on the status of these copies to each other. We call requests where servers perform cross-server status updates “tied requests.” The simplest form of a tied request has the client send the request to two different servers, each tagged with the identity of the other server (“tied”). When a request begins execution, it sends a cancellation message to its counterpart. The corresponding request, if still enqueued in the other server, can be aborted immediately or deprioritized substantially.

There is a brief window of one average network message delay where both servers may start executing the request while the cancellation messages are both in flight to the other server. A common case where this situation can occur is if both server queues are completely empty. It is useful therefore for the client to introduce a small delay of two times the average network message delay (1ms or less in modern data-center networks) between sending the first request and sending the second request.

Google’s implementation of this technique in the context of its cluster-level distributed file system is effective at reducing both median and tail latencies. Table 2 lists the times for servicing a small read request from a BigTable where the data is not cached in memory but must be read from the underlying file system; each file chunk has three replicas on distinct machines. The table includes read latencies observed with and without tied requests for two scenarios: The first is a cluster in which the benchmark is running in isolation, in which case latency variability is mostly

from self-interference and regular cluster-management activities. In it, sending a tied request that does cross-server cancellation to another file system replica following 1ms reduces median latency by 16% and is increasingly effective along the tail of the latency distribution, achieving nearly 40% reduction at the 99.9th-percentile latency. The second scenario is like the first except there is also a large, concurrent sorting job running on the same cluster contending for the same disk resources in the shared file system. Although overall latencies are somewhat higher due to higher utilization, similar reductions in the latency profile are achieved with the tied-request technique discussed earlier. The latency profile with tied requests while running a concurrent large sorting job is nearly identical to the latency profile of a mostly idle cluster without tied requests. Tied requests allow the workloads to be consolidated into a single cluster, resulting in dramatic computing cost reductions. In both Table 2 scenarios, the overhead of tied requests in disk utilization is less than 1%, indicating the cancellation strategy is effective at eliminating redundant reads.

An alternative to the tied-request and hedged-request schemes is to probe remote queues first, then submit the request to the least-loaded server.¹⁰ It can be beneficial but is less effective than submitting work to two queues simultaneously for three main reasons: load levels can change between probe and request time; request service times can be difficult to estimate due to underlying system and hardware variability; and clients can create temporary hot spots by all clients picking the same (least-loaded) server at the same time. The

Distributed Shortest-Positioning Time First system⁹ uses another variation in which the request is sent to one server and forwarded to replicas only if the initial server does not have it in its cache and uses cross-server cancellations.

Worth noting is this technique is not restricted to simple replication but is also applicable in more-complex coding schemes (such as Reed-Solomon) where a primary request is sent to the machine with the desired data block, and, if no response is received following a brief delay, a collection of requests is issued to a subset of the remaining replication group sufficient to reconstruct the desired data, with the whole ensemble forming a set of tied requests.

Note, too, the class of techniques described here is effective only when the phenomena that causes variability does not tend to simultaneously affect multiple request replicas. We expect such uncorrelated phenomena are rather common in large-scale systems.

Cross-Request Long-Term Adaptations

Here, we turn to techniques that are applicable for reducing latency variability caused by coarser-grain phenomena (such as service-time variations and load imbalance). Although many systems try to partition data in such a way that the partitions have equal cost, a static assignment of a single partition to each machine is rarely sufficient in practice for two reasons: First, the performance of the underlying machines is neither uniform nor constant over time, for reasons (such as thermal throttling and shared workload interference) mentioned earlier. And second, outliers in the assignment of items to partitions can cause data-induced load imbalance (such as when a particular item becomes popular and the load for its partition increases).

Micro-partitions. To combat imbalance, many of Google’s systems generate many more partitions than there are machines in the service, then do dynamic assignment and load balancing of these partitions to particular machines. Load balancing is then a matter of moving responsibility for one of these small partitions from one machine to another. With an average of, say, 20 partitions per machine, the system can shed load in roughly 5% increments and


Table 2. Read latencies observed in a BigTable service benchmark.

	Mostly idle cluster			With concurrent terasort		
	No hedge	Tied request after 1ms		No hedge	Tied request after 1ms	
50%ile	19ms	16ms	(-16%)	24ms	19ms	(-21%)
90%ile	38ms	29ms	(-24%)	56ms	38ms	(-32%)
99%ile	67ms	42ms	(-37%)	108ms	67ms	(-38%)
99.9%ile	98ms	61ms	(-38%)	159ms	108ms	(-32%)


in 1/20th the time it would take if the system simply had a one-to-one mapping of partitions to machines. The BigTable distributed-storage system stores data in tablets, with each machine managing between 20 and 1,000 tablets at a time. Failure-recovery speed is also improved through micro-partitioning, since many machines pick up one unit of work when a machine failure occurs. This method of using micro-partitions is similar to the virtual servers notion as described in Stoica¹² and the virtual-processor-partitioning technique in DeWitt et al.⁶

Selective replication. An enhancement of the micro-partitioning scheme is to detect or even predict certain items that are likely to cause load imbalance and create additional replicas of these items. Load-balancing systems can then use the additional replicas to spread the load of these hot micro-partitions across multiple machines without having to actually move micro-partitions. Google's main Web search system uses this approach, making additional copies of popular and important documents in multiple micro-partitions. At various times in Google's Web search system's evolution, it has also created micro-partitions biased toward particular document languages and adjusted replication of these micro-partitions as the mix of query languages changes through the course of a typical day. Query mixes can also change abruptly, as when, say, an Asian data-center outage causes a large fraction of Asian-language queries to be directed to a North American facility, materially changing its workload behavior.

Latency-induced probation. By observing the latency distribution of responses from the various machines in the system, intermediate servers sometimes detect situations where the system performs better by excluding a particularly slow machine, or putting it on probation. The source of the slowness is frequently temporary phenomena like interference from unrelated networking traffic or a spike in CPU activity for another job on the machine, and the slowness tends to be noticed when the system is under greater load. However, the system continues to issue shadow requests to these excluded servers, collecting statistics on their latency so they can be reincorporated into the service when the problem abates. This situa-



A simple way to curb latency variability is to issue the same request to multiple replicas and use the results from whichever replica responds first.



tion is somewhat peculiar, as removal of serving capacity from a live system during periods of high load actually improves latency.

Large Information Retrieval Systems

In large information-retrieval (IR) systems, speed is more than a performance metric; it is a key quality metric, as returning good results quickly is better than returning the best results slowly. Two techniques apply to such systems, as well as other to systems that inherently deal with imprecise results:

Good enough. In large IR systems, once a sufficient fraction of all the leaf servers has responded, the user may be best served by being given slightly incomplete (“good-enough”) results in exchange for better end-to-end latency. The chance that a particular leaf server has the best result for the query is less than one in 1,000 queries, odds further reduced by replicating the most important documents in the corpus into multiple leaf servers. Since waiting for exceedingly slow servers might stretch service latency to unacceptable levels, Google's IR systems are tuned to occasionally respond with good-enough results when an acceptable fraction of the overall corpus has been searched, while being careful to ensure good-enough results remain rare. In general, good-enough schemes are also used to skip nonessential subsystems to improve responsiveness; for example, results from ads or spelling-correction systems are easily skipped for Web searches if they do not respond in time.

Canary requests. Another problem that can occur in systems with very high fan-out is that a particular request exercises an untested code path, causing crashes or extremely long delays on thousands of servers simultaneously. To prevent such correlated crash scenarios, some of Google's IR systems employ a technique called “canary requests”; rather than initially send a request to thousands of leaf servers, a root server sends it first to one or two leaf servers. The remaining servers are only queried if the root gets a successful response from the canary in a reasonable period of time. If the server crashes or hangs while the canary request is outstanding, the system flags the request as potentially dangerous and prevents further ex-

ecution by not sending it to the remaining leaf servers. Canary requests provide a measure of robustness to back-ends in the face of difficult-to-predict programming errors, as well as malicious denial-of-service attacks.

The canary-request phase adds only a small amount of overall latency because the system must wait for only a single server to respond, producing much less variability than if it had to wait for all servers to respond for large fan-out requests; compare the first and last rows in Table 1. Despite the slight increase in latency caused by canary requests, such requests tend to be used for every request in all of Google's large fan-out search systems due to the additional safety they provide.

Mutations

The techniques we have discussed so far are most applicable for operations that do not perform critical mutations of the system's state, which covers a broad range of data-intensive services. Tolerating latency variability for operations that mutate state is somewhat easier for a number of reasons: First, the scale of latency-critical modifications in these services is generally small. Second, updates can often be performed off the critical path, after responding to the user. Third, many services can be structured to tolerate inconsistent update models for (inherently more latency-tolerant) mutations. And, finally, for those services that require consistent updates, the most commonly used techniques are quorum-based algorithms (such as Lamport's Paxos⁸); since these algorithms must commit to only three to five replicas, they are inherently tail-tolerant.

Hardware Trends and Their Effects

Variability at the hardware level is likely to be higher in the future due to more aggressive power optimizations becoming available and fabrication challenges at deep submicron levels resulting in device-level heterogeneity. Device heterogeneity combined with ever-increasing system scale will make tolerating variability through software techniques even more important over time. Fortunately, several emerging hardware trends will increase the effectiveness of latency-tolerating techniques. For example, higher bisection bandwidth

in data-center networks and network-interface optimizations that reduce per-message overheads (such as remote direct-memory access) will reduce the cost of tied requests, making it more likely that cancellation messages are received in time to avoid redundant work. Lower per-message overheads naturally allow more fine-grain requests, contributing to better multiplexing and avoiding head-of-line blocking effects.

Conclusion

Delivering the next generation of compute-intensive, seamlessly interactive cloud services requires consistently responsive massive-scale computing systems that are only now beginning to be contemplated. As systems scale up, simply stamping out all sources of performance variability will not achieve such responsiveness. Fault-tolerant techniques were developed because guaranteeing fault-free operation became infeasible beyond certain levels of system complexity. Similarly, tail-tolerant techniques are being developed for large-scale services because eliminating all sources of variability is also infeasible. Although approaches that address particular sources of latency variability are useful, the most powerful tail-tolerant techniques reduce latency hiccups regardless of root cause. These tail-tolerant techniques allow designers to continue to optimize for the common case while providing resilience against uncommon cases. We have outlined a small collection of tail-tolerant techniques that have been effective in several of Google's large-scale software systems. Their importance will only increase as Internet services demand ever-larger and more complex warehouse-scale systems and as the underlying hardware components display greater performance variability.

While some of the most powerful tail-tolerant techniques require additional resources, their overhead can be rather modest, often relying on existing capacity already provisioned for fault-tolerance while yielding substantial latency improvements. In addition, many of these techniques can be encapsulated within baseline libraries and systems, and the latency improvements often enable radically simpler application-level designs. Besides enabling low

latency at large scale, these techniques make it possible to achieve higher system utilization without sacrificing service responsiveness.

Acknowledgments

We thank Ben Appleton, Zhifeng Chen, Greg Ganger, Sanjay Ghemawat, Ali Ghodsi, Rama Govindaraju, Lawrence Greenfield, Steve Gribble, Brian Gustafson, Nevin Heintze, Jeff Mogul, Andrew Moore, Rob Pike, Sean Quinlan, Gautham Thambidorai, Ion Stoica, Amin Vahdat, and T.N. Vijaykumar for their helpful feedback on earlier drafts and presentations of this work. Numerous people at Google have worked on systems that use these techniques. □

References

1. Barroso, L.A. and Hölzle, U. The case for energy proportional computing. *IEEE Computer* 40, 12 (Dec. 2007), 33–37.
2. Barroso, L.A. and Hölzle, U. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-scale Machines*. Synthesis Series on Computer Architecture, Morgan & Claypool Publishers, May 2009.
3. Card, S.K., Robertson, G.G., and Mackinlay, J.D. The information visualizer: An information workspace. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems* (New Orleans, Aug. 28–May 2). ACM Press, New York, 1991, 181–188.
4. Chang F., Dean J., Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M., Chandra, T., Fikes, A., and Gruber, R.E. BigTable: A distributed storage system for structured data. In *Proceedings of the Seventh Symposium on Operating Systems Design and Implementation* (Seattle, Nov.). USENIX Association, Berkeley CA, 2006, 205–218.
5. Charles, J., Jassi, P., Ananth, N.S., Sadat, A., and Fedorova, A. Evaluation of the Intel Core i7 Turbo Boost feature. In *Proceedings of the IEEE International Symposium on Workload Characterization* (Austin, TX, Oct. 4–6). IEEE Computer Society Press, 2009, 188–197.
6. DeWitt, D.J., Naughton, J.F., Schneider, D.A., and Seshadri, S. Practical skew handling in parallel joins. In *Proceedings of the 18th International Conference on Very Large Data Bases*, Li-Yan Yuan, Ed. (Vancouver, BC, Aug. 24–27). Morgan Kaufmann Publishers, Inc., San Francisco, 1992, 27–40.
7. Google, Inc. Project Glass; <http://g.co/projectglass>
8. Lamport, L. The part-time parliament. *ACM Transactions on Computer Systems* 16, 2 (May 1998), 133–169.
9. Lumb, C.R. and Golding, R. D-SPTF: Decentralized request distribution in brick-based storage systems. *SIGOPS Operating System Review* 38, 5 (Oct. 2004), 37–47.
10. Mitzenmacher, M. The power of two choices in randomized load balancing. *IEEE Transactions on Parallel and Distributed Computing* 12, 10 (Oct. 2001), 1094–1104.
11. Mudge, T. and Hölzle, U. Challenges and opportunities for extremely energy-efficient processors. *IEEE Micro* 30, 4 (July 2010), 20–24.
12. Stoica I., Morris, R., Karger, D., Kaashoek, F., and Balakrishnan, H. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proceedings of SIGCOMM* (San Diego, Aug. 27–31). ACM Press, New York, 2001, 149–160.

Jeffrey Dean (jeff@google.com) is a Google Fellow in the Systems Infrastructure Group of Google Inc., Mountain View, CA.

Luiz André Barroso (luiz@google.com) is a Google Fellow and technical lead of core computing infrastructure at Google Inc., Mountain View, CA.

ACM TechNews Goes Mobile

iPhone & iPad Apps Now Available in the iTunes Store

ACM TechNews—ACM's popular thrice-weekly news briefing service—is now available as an easy to use mobile apps downloadable from the Apple iTunes Store.

These new apps allow nearly 100,000 ACM members to keep current with news, trends, and timely information impacting the global IT and Computing communities each day.



TechNews mobile app users will enjoy:

- **Latest News:** Concise summaries of the most relevant news impacting the computing world
- **Original Sources:** Links to the full-length articles published in over 3,000 news sources
- **Archive access:** Access to the complete archive of TechNews issues dating back to the first issue published in December 1999
- **Article Sharing:** The ability to share news with friends and colleagues via email, text messaging, and popular social networking sites
- **Touch Screen Navigation:** Find news articles quickly and easily with a streamlined, fingertip scroll bar
- **Search:** Simple search the entire TechNews archive by keyword, author, or title
- **Save:** One-click saving of latest news or archived summaries in a personal binder for easy access
- **Automatic Updates:** By entering and saving your ACM Web Account login information, the apps will automatically update with the latest issues of TechNews published every Monday, Wednesday, and Friday

The Apps are freely available to download from the Apple iTunes Store, but users must be registered individual members of ACM with valid Web Accounts to receive regularly updated content.

<http://www.apple.com/iphone/apps-for-iphone/> <http://www.apple.com/ipad/apps-for-ipad/>

ACM TechNews



**The challenges—and great promise—
of modern symbolic execution techniques,
and the tools to help implement them.**

BY CRISTIAN CADAR AND KOUSHIK SEN

Symbolic Execution for Software Testing: Three Decades Later

SYMBOLIC EXECUTION HAS garnered a lot of attention in recent years as an effective technique for generating high-coverage test suites and for finding deep errors in complex software applications. While the key idea behind symbolic execution was introduced more than three decades ago,^{6,12,23} it has only recently been made practical, as a result of significant advances in constraint satisfiability,¹⁶ and of more scalable dynamic approaches that combine concrete and symbolic execution.^{9,19}

Symbolic execution is typically used in software testing to explore as many different program paths as possible in a given amount of time, and for each path to generate a set of concrete input values exercising it, and

check for the presence of various kinds of errors including assertion violations, uncaught exceptions, security vulnerabilities, and memory corruption. The ability to generate concrete test inputs is one of the major strengths of symbolic execution: from a test generation perspective, it allows the creation of high-coverage test suites, while from a bug-finding perspective, it provides developers with a concrete input that triggers the bug, which can be used to confirm the error independently of the symbolic execution tool that generated it.

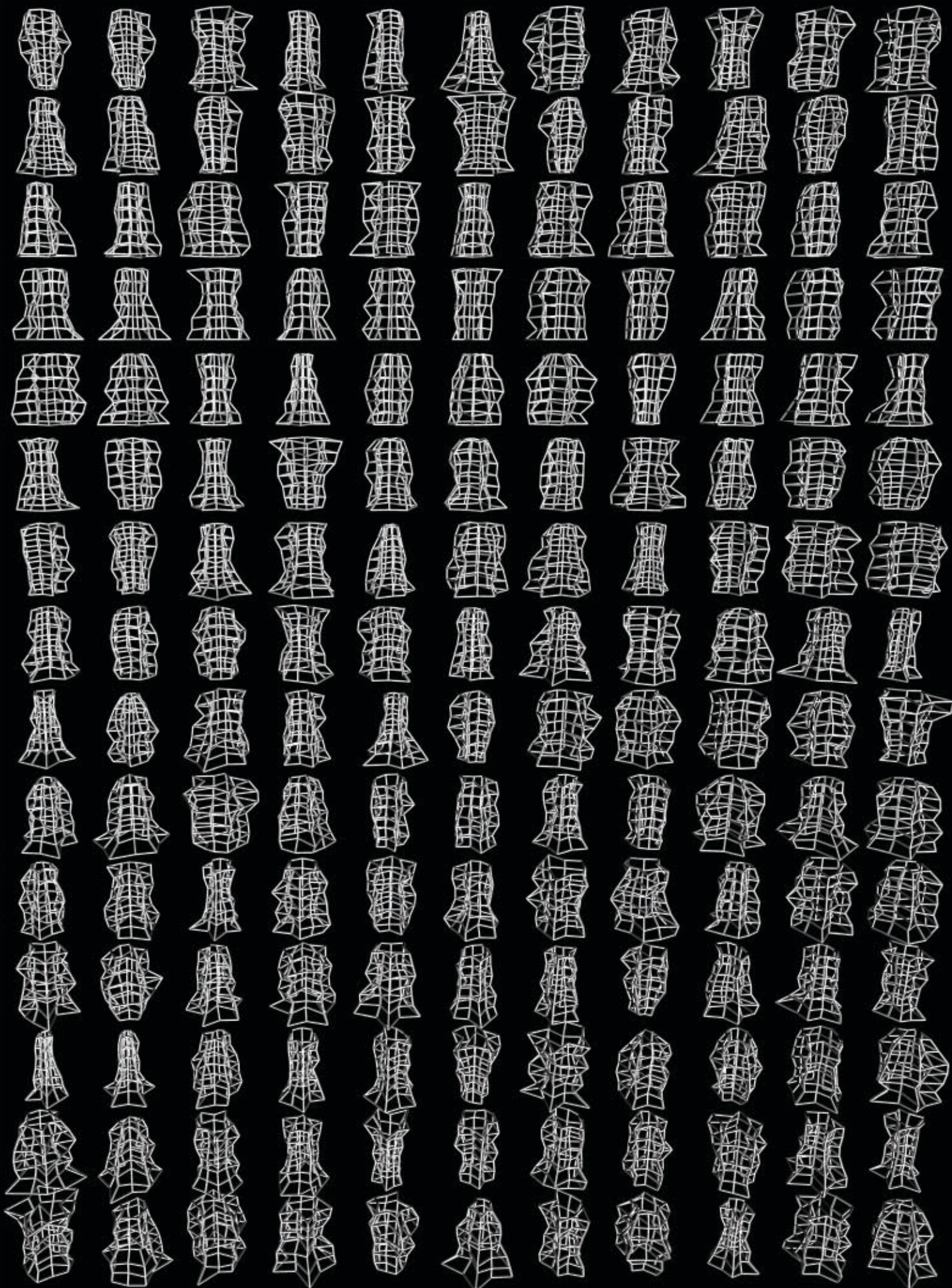
Furthermore, note that in terms of finding errors on a given program path, symbolic execution is much more powerful than traditional dynamic execution techniques such as those implemented by popular tools like Valgrind²⁸ or Purify,²¹ because it has the ability to find a bug if there are *any* buggy inputs on that path, rather than depending on having a concrete input that triggers the bug.

Finally, unlike other program analysis techniques, symbolic execution is not limited to finding generic errors such as buffer overflows, but can reason about higher-level program properties, such as complex program assertions.

This article gives an overview of symbolic execution by showing how it

» key insights

- **Modern symbolic execution techniques provide an effective way to automatically generate test inputs for real-world software. Such inputs can achieve high test coverage and find corner-case bugs such as buffer overflows, uncaught exceptions, and assertion violations.**
- **Symbolic execution works by exploring as many program paths as possible in a given time budget, creating logical formula encoding the explored paths, and using a constraint solver to generate test inputs for feasible execution paths.**
- **Modern symbolic execution techniques mix concrete and symbolic execution and benefit from significant advances in constraint solving to alleviate limitations which prevented traditional symbolic execution from being useful in practice for about 30 years.**



works on a simple example and highlighting its main features. We describe a couple of modern approaches to symbolic execution that make it effective for real-world software. Then, we explore the main challenges of symbolic execution, including path explosion, constraint solving, and memory modeling. Finally, we present several representative symbolic execution tools. Note that we do not aim to provide here a comprehensive survey of existing work in the area, but instead choose to illustrate some of the main challenges and proposed solutions by using examples from the authors' own work.

Overview of Classical Symbolic Execution

The key idea behind symbolic execution^{6,12,23} is to use *symbolic values*, instead of concrete data values, as input values, and to represent the values of program variables as *symbolic expressions* over the symbolic values. As a result, the output values computed by a program are expressed as a function of the input symbolic values. In software testing, symbolic execution is used to generate a test input for each feasible execution path of a program. A feasible execution path is a sequence of `true` and `false`, where a value of `true` (respectively `false`) at the i^{th} position in the sequence denotes that the i^{th} conditional statement encountered along the execution path took the “then” (respectively the “else”) branch. All the feasible execution paths of a program can be represented using a tree, called the *execution tree*. For example, the function `testme()` in Figure 1 has three feasible execution paths, which form the execution tree shown in Figure 2. These paths can be executed, for instance, by running the program on the inputs $\{x = 0, y = 1\}$, $\{x = 2, y = 1\}$ and $\{x = 30, y = 15\}$. The goal of symbolic execution is to generate such a set of inputs so that all the feasible execution paths (or as many as possible in a given time budget) can be explored exactly once by running the program on those inputs.

Symbolic execution maintains a symbolic state σ , which maps variables to symbolic expressions, and a symbolic path constraint (or path condition) PC , which is a quantifier-free first-order formula over symbolic expressions. At the beginning of a symbolic execution,

Unlike other program analysis techniques, symbolic execution is not limited to finding generic errors such as buffer overflows, but can reason about higher-level program properties, such as complex program assertions.

σ is initialized to an empty map and PC is initialized to `true`. Both σ and PC are populated during the course of symbolic execution. At the end of a symbolic execution along a feasible execution path of the program, PC is solved using a constraint solver to generate concrete input values. If the program is executed on these concrete input values, it will take exactly the same path as the symbolic execution and terminate in the same way.

For example, symbolic execution of the code in Figure 1 starts with an empty symbolic state and with symbolic path constraint `true`. At every read statement `var = sym_input()` that receives program input, symbolic execution adds the mapping $var \mapsto s$ to σ , where s is a fresh symbolic value. For example, symbolic execution of the first two lines of the `main()` function (lines 16–17) results in $\sigma = \{x \mapsto x_0, y \mapsto y_0\}$, where x_0, y_0 are two initially unconstrained symbolic values. At every assignment $v = e$, symbolic execution updates σ by mapping v to $\sigma(e)$, the symbolic expression obtained by evaluating e in the current symbolic state. For example, after executing line 6, $\sigma = \{x \mapsto x_0, y \mapsto y_0, z \mapsto 2y_0\}$.

At every conditional statement `if (e) S1 else S2`, PC is updated to $PC \wedge \sigma(e)$ (“then” branch), and a fresh path constraint PC' is created and initialized to $PC \wedge \neg \sigma(e)$ (“else” branch). If PC is satisfiable for some assignment of concrete to symbolic values, then symbolic execution continues along the “then” branch with the symbolic state σ and symbolic path constraint PC . Similarly, if PC' is satisfiable, then another instance of symbolic execution is created with symbolic state σ and symbolic path constraint PC' , which continues the execution along the “else” branch; note that unlike in concrete execution, both branches can be taken, resulting in two execution paths. If any of PC or PC' is not satisfiable, symbolic execution terminates along the corresponding path. For example, after line 7 in the example code, two instances of symbolic execution are created with path constraints $x_0 = 2y_0$ and $x_0 \neq 2y_0$, respectively. Similarly, after line 8, two instances of symbolic execution are created with path constraints $(x_0 = 2y_0) \wedge (x_0 > y_0 + 10)$ and $(x_0 = 2y_0) \wedge (x_0 \leq y_0 + 10)$, respectively.

If a symbolic execution instance hits an exit statement or an error (for example, the program crashes or violates an assertion), the current instance of symbolic execution is terminated and a satisfying assignment to the current symbolic path constraint is generated, using an off-the-shelf constraint solver. The satisfying assignment forms the *test inputs*: if the program is executed on these concrete input values, it will take exactly the same path as the symbolic execution and terminate in the same way. For example, on our example code we get three instances of symbolic executions that result in the test inputs $\{x = 0, y = 1\}$, $\{x = 2, y = 1\}$, and $\{x = 30, y = 15\}$, respectively.

Symbolic execution of code containing loops or recursion may result in an infinite number of paths if the termination condition for the loop or recursion is symbolic. For example, the code in Figure 3 has an infinite number of feasible execution paths, where each feasible execution path is either a sequence of an arbitrary number of true's followed by a false or a sequence of infinite number of true's. The symbolic path constraint of a path with a sequence of n true's followed by a false is:

$$\left(\bigwedge_{i \in [1, n]} N_i > 0\right) \wedge (N_{n+1} \leq 0)$$

where each N_i is a fresh symbolic value, and the symbolic state at the end of the execution is $\{N \mapsto N_{n+1}, \text{sum} \mapsto \sum_{i \in [1, n]} N_i\}$. In practice, one needs to put a limit on the search (for example, a timeout, or a limit on the number of paths, loop iterations, or exploration depth).

A key disadvantage of classical symbolic execution is that it cannot generate an input if the symbolic path constraint along a feasible execution path contains formulas that cannot be (efficiently) solved by a constraint solver (for example, nonlinear constraints). Consider performing symbolic execution on two variants of the code in Figure 1: in one variant, we modify the `twice` function as in Figure 4; in the other variant, we assume that the code of `twice` is not available. Let us assume that our constraint solver cannot handle non-linear arithmetic. For the first variant, symbolic execution will generate the path constraints $x_0 \neq (y_0 y_0) \% 50$ and $x_0 = (y_0 y_0) \% 50$ after the execution

Figure 1. Simple example illustrating symbolic execution.

```

1  int twice (int v) {
2      return 2*v;
3  }
4
5  void testme (int x, int y) {
6      z = twice (y);
7      if (z == x) {
8          if (x > y+10)
9              ERROR;
10         }
11     }
12 }
13
14 /* simple driver exercising testme () with sym inputs */
15 int main() {
16     x = sym_input ();
17     y = sym_input ();
18     testme (x, y);
19     return 0;
20 }

```

of the first conditional statement. For the second variant, symbolic execution will generate the path constraints $x_0 \neq \text{twice}(y_0)$ and $x_0 = \text{twice}(y_0)$, where `twice` is an uninterpreted function. Since the constraint solver cannot solve any of these constraints, symbolic execution will fail to generate any input for the modified programs. We next describe two modern symbolic execution techniques that alleviate this problem and generate at least some inputs for the modified programs.

Modern Symbolic Execution Techniques

One of the key elements of modern symbolic execution techniques is their ability to mix concrete and symbolic execution. We present here two such extensions, and then discuss the main advantages they provide.

Concolic Testing. Directed Automated Random Testing (DART)¹⁹ or concolic testing³⁵ performs symbolic execution dynamically, while the program is executed on some concrete input values. Concolic testing maintains a concrete state and a symbolic state: the concrete state maps all variables to their concrete values; the symbolic state only maps variables that have non-concrete values. Unlike classical symbolic execution, since concolic execution maintains the entire concrete state of the program along an execution, it needs initial concrete values for its inputs. Concolic testing executes a program starting with some given or

Figure 2. Execution tree for the example in Figure 1.

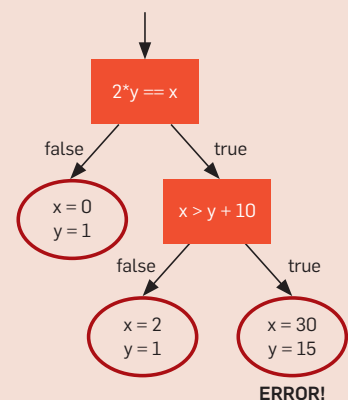


Figure 3. Simple example illustrating an infinite number of feasible execution paths.

```

1  void testme_inf() {
2      int sum = 0;
3      int N = sym_input ();
4      while (N > 0) {
5          sum = sum + N
6          N = sym_input ();
7      }
8  }

```

Figure 4. Simple modification of the example in Figure 1. The function `twice` now performs some non-linear computation.

```

1  int twice (int v) {
2      return (v*v) % 50;
3  }

```

random input, gathers symbolic constraints on inputs at conditional statements along the execution, and then uses a constraint solver to infer variants of the previous inputs in order to steer the next execution of the program toward an alternative feasible execution path. This process is repeated systematically or heuristically until all feasible execution paths are explored or a user-defined coverage criteria is met.

For the example in Figure 1, concolic execution will generate some random input, say $\{x = 22, y = 7\}$, and execute the program both concretely and symbolically. The concrete execution will take the “else” branch at line 7 and the symbolic execution will generate the path constraint $x_0 \neq 2y_0$ along the concrete execution path. Concolic testing negates a conjunct in the path constraint and solves $x_0 = 2y_0$ to get the test input $\{x = 2, y = 1\}$; this new input will force the program execution along a different execution path. Concolic testing repeats both concrete and symbolic execution on this new test input. The execution takes a path different from the previous one—the “then” branch at line 7 and the “else” branch at line 8 are now taken in this execution. As in the previous execution, concolic testing also performs symbolic execution along this concrete execution and generates the path constraint $(x_0 = 2y_0) \wedge (x_0 \leq y_0 + 10)$. Concolic testing will generate a new test input that forces the program along an execution path that has not been previously executed. It does so by negating the conjunct $(x_0 \leq y_0 + 10)$ and solving the constraint $(x_0 = 2y_0) \wedge (x_0 > y_0 + 10)$ to get the test input $\{x = 30, y = 15\}$. The program reaches the ERROR statement with this new input. After this third execution of the program, concolic testing reports that all execution paths of the program have been explored and terminates test input generation. Note that in this example, concolic testing explores all the execution paths using a depth-first search strategy; however, one could employ other strategies to explore paths in different orders, as we discuss later.

Execution-Generated Testing (EGT). The EGT approach,⁹ implemented and extended by the EXE¹⁰ and KLEE⁸ tools, works by making a distinction between the concrete and symbolic state of a program. To this end, EGT intermixes

concrete and symbolic execution by dynamically checking before every operation if the values involved are all concrete. If so, the operation is executed just as in the original program. Otherwise, if at least one value is symbolic, the operation is performed symbolically, by updating the path condition for the current path. For example, if line 17 in Figure 1 is changed to $y = 10$, then line 6 will simply call function `twice()` with the concrete argument 10, call which will be executed as in the original program (note that `twice` could perform an arbitrarily complex operation on its input, but this would not place any additional strain on symbolic execution, because the call will be executed concretely). Then, the branch on line 7 will become `if (20 == x)`, and execution will be forked, one instance adding the constraint that $x = 20$ and following the “then” branch, and the other adding the constraint that $x \neq 20$ and following the “else” branch. Note that on the “then” branch, the conditional at line 8 becomes `if (x > 20)`, and therefore its “then” side is infeasible because x is constrained to have value 20 on this path.

Imprecision vs. completeness in concolic testing and EGT. One of the key advantages in mixing concrete and symbolic execution is that imprecision in symbolic execution (due to, for example, interaction with external code, or constraint solving timeouts), can be alleviated using concrete values (and in the case of concolic testing, also randomization).

For example, real applications almost always interact with the outside world, for instance, by calling libraries that are not instrumented for symbolic execution, or by issuing OS system calls. If all the arguments passed to such a call are concrete, the call can be simply performed concretely, as in the original program. However, even if some operands are symbolic, EGT and concolic testing can use one of the possible concrete values of the symbolic arguments: in EGT this is done by solving the current path constraint for a satisfying assignment, while concolic testing can immediately use the concrete runtime values of those inputs from the current concolic execution.

Besides external code, imprecision in symbolic execution creeps into many

other places—such as unhandled instructions (for example, floating-point) or complex functions that cause constraint solver timeouts—and the use of concrete values allows symbolic execution to recover from that imprecision, albeit at the cost of missing some feasible paths, and thus sacrificing completeness.

To illustrate, we describe the behavior of concolic testing on the version of our running example in which the function `twice` returns the non-linear value $(v*v)\%50$ (see Figure 4). Let us assume that concolic testing generates the random input $\{x = 22, y = 7\}$. Then, the symbolic execution will generate the symbolic path constraint $x_0 \neq (y_0 y_0)\%50$ along the concrete execution path on this input. If we assume that the constraint solver cannot handle non-linear constraints, then concolic testing will fail to generate an input for an alternate execution path. We get a similar situation if the source code for the function `twice` is not available (for example, `twice` is some third-party closed-source library function or a system call), in which case the path constraint becomes $x_0 \neq \text{twice}(y_0)$, where `twice` is an uninterpreted function. Concolic testing handles this situation by replacing some of the symbolic values with their concrete values so that the resultant constraints are simplified and can be solved. For instance, in the example, concolic testing replaces y_0 by its concrete value 7. This simplifies the path constraint in both program versions to $x_0 \neq 49$. By solving the path constraint $x_0 = 49$, concolic testing generates the new input $\{x = 49, y = 7\}$ for a previously unexplored execution path. Note that classical symbolic execution cannot perform this simplification because the concrete state is not available during symbolic execution.

EGT can handle this situation in a similar way: when it encounters the statement `return (v*v) % 50` or the external call `z = twice(y)`, it will call the constraint solver on the current symbolic path constraint to generate a satisfying assignment to y_0 , say $y_0 = 7$, replace this value in the symbolic state and in the path constraint, and continue the execution in a partial symbolic state $\{x \mapsto x_0, y \mapsto 7\}$. The tool KLEE optimizes this by keeping a counterexample cache (described later).

Concolic testing and EGT's approach to simplify constraints using concrete values help them generate test inputs for execution paths for which symbolic execution gets stuck, but this approach comes with a caveat: due to simplification, concolic testing and EGT could lose completeness, that is, they may not be able to generate test inputs for some feasible execution paths. For instance, in our example both techniques will fail to generate an input for the path `true, false`. However, this is clearly preferable to the alternative of simply aborting execution when unsupported statements or external calls are encountered.

Challenges and Extensions

Here, we discuss the main challenges in symbolic execution, and some interesting solutions and extensions developed in response to them.

Path Explosion. One of the key challenges of symbolic execution is the huge number of program paths in all but the smallest programs, which is usually exponential in the number of static branches in the code. However, note that symbolic execution explores only feasible paths that depend on the symbolic input, which reduces the number of conditionals that spawn new paths. For example, in several experiments on testing a number of medium-sized applications we found that less than 42% of the executed statements depend on the symbolic input, and often less than 20% of the symbolic branches encountered during execution have both sides feasible.¹⁰

Despite this implicit filtering, path explosion represents one of the biggest challenges facing symbolic execution, and given a fixed time budget, it is critical to explore the most relevant paths first. Here, we present a representative selection of the techniques developed to address this problem.

Search heuristics. The main mechanism used by symbolic execution tools to prioritize path exploration is the use of search heuristics. Most heuristics focus on achieving high statement and branch coverage, but they could also be employed to optimize other desired criteria. We describe here several coverage-optimized search heuristics successfully used by current symbolic

Path explosion represents one of the biggest challenges facing symbolic execution.

execution tools.

One particularly effective approach is to use the static control-flow graph (CFG) to guide the exploration toward the path closest (as measured statically using the CFG) from an uncovered instruction.^{7,8} A similar approach, described in Cadar et al.,¹⁰ is to favor statements that were run the fewest number of times.

As another example, heuristics based on random exploration have also proved successful.^{7,8} The main idea is to start from the beginning of the program, and at each symbolic branch for which both sides are feasible to randomly choose which side to explore. Note that this random strategy has a number of important advantages: compared to randomly choosing a path to execute, it avoids starvation when a part of the program rapidly forks many new paths; and compared to randomly generating inputs, it has a higher probability to reach branches that are covered by a very small fraction of the inputs. Furthermore, this strategy favors paths early in the execution, with fewer constraints on the inputs, and thus on reaching new program statements.

Interleaving random and symbolic execution. Another successful approach, which was explored in the context of concolic testing, is to interleave symbolic exploration with random testing.²⁶ This approach combines the ability of random testing to quickly reach deep execution states, with the power of symbolic execution to thoroughly explore states in a given neighborhood.

Pruning redundant paths. An alternative approach to avoid exploring the same lines of code over and over again is to automatically prune redundant paths during exploration. The key insight behind the RWset technique described in Boonstoppel et al.⁵ is that if a program path reaches the same program point with the same symbolic constraints as a previously explored path, then this path will continue to execute exactly the same from that point on and thus can be discarded. This technique is enhanced by an important optimization: when comparing the constraints on the two execution paths, it discards those that depend only on values that will not be subsequently read by the program. Note that the effect of pruning these paths can be


significant, as the number of new paths spawned by the continued execution can be exponential in the number of encountered branches.

Lazy test generation. Lazy test generation²⁷ is an approach similar to the counterexample-guided refinement paradigm from static software verification. The technique first explores, using concolic execution, an abstraction of the function under test by replacing each called function with an unconstrained input. Second, for each (possibly spurious) trace generated by this abstraction, it attempts to expand the trace to a concretely realizable execution by recursively expanding the called functions and finding concrete executions in the called functions that can be stitched together with the original trace to form a complete program execution. Thus, it reduces the burden of symbolic reasoning about interprocedural paths to reasoning about intraprocedural paths (in the exploration phase), together with a localized and constrained search through functions (in the concretization phase).


Static path merging. One simple approach that can be used to reduce the number of paths explored is to merge them statically using select expressions that are then passed directly to the constraint solver.¹³ For example, the statement $x[i] = x[i] > 0 ? x[i] : -x[i]$ can be encoded as $(x[i] = \text{select}(x[i] > 0, x[i], -x[i]))$. If such an expression is computed inside a loop statement with N iterations, this approach can reduce the number of explored paths from 2^N to 1. While merging can be effective in many cases, it is unfortunately passing the complexity to the constraint solver, which as discussed in the next section represents another major challenge of symbolic execution.

Constraint Solving. Despite significant advances in constraint solving technology during the last few years—which made symbolic execution practical in the first place—constraint solving continues to be one of the main bottlenecks in symbolic execution, where it often dominates runtime. In fact, one of the main reasons for which symbolic execution fails to scale on some programs is that their code is generating queries that are blowing up the solver.

As a result, it is essential to implement constraint-solving optimizations



It is essential to implement constraint-solving optimizations that exploit the type of constraints generated during the symbolic execution of real programs.



that exploit the type of constraints generated during the symbolic execution of real programs. We present here two representative optimizations used by existing symbolic execution tools.

Irrelevant constraint elimination. The vast majority of queries in symbolic execution are issued in order to determine the feasibility of taking a certain branch side. For example, in the concolic variant of symbolic execution, one branch predicate of an existing path constraint is negated and then the resulting constraint set is checked for satisfiability in order to determine if the program can take the other side of the branch, corresponding to the negated constraint. An important observation is that in general a program branch depends only on a small number of program variables, and therefore on a small number of constraints from the path condition. Thus, one effective optimization is to remove from the path condition those constraints that are irrelevant in deciding the outcome of the current branch. For example, let the path condition be $(x + y > 10) \wedge (z > 0) \wedge (y < 12) \wedge (z - x = 0)$ and suppose we want to generate a new input by solving $(x + y > 10) \wedge (z > 0) \wedge \neg(y < 12)$, where $\neg(y < 12)$ is the negated branch condition whose feasibility we are trying to establish. Then it is safe to eliminate the constraint on z , because this constraint cannot influence the outcome of the $y < 12$ branch. The solution of this reduced constraint set will give new values for x and y , and we use the value of z from the current execution to generate the new input. More formally, the algorithm computes the transitive closure of all the constraints on which the negated constraint depends, by looking whether they share any variables between them. The extra complication is in dealing with pointer dereferences and array indexing, which is discussed in detail in Cadar et al.¹⁰ and Sen et al.³⁵

Incremental solving. One important characteristic of the constraint sets generated during symbolic execution is that they are expressed in terms of a fixed set of static branches from the program source code. For this reason, many paths have similar constraint sets, and thus allow for similar solutions; this fact can be exploited to improve the speed of constraint solving

by reusing the results of previous similar queries, as done in several systems such as CUTE and KLEE.^{8,35} To illustrate this point, we present one such algorithm, namely the counterexample caching scheme used by KLEE.⁸ In KLEE, all query results are stored in a cache that maps constraint sets to concrete variable assignments (or a special *No solution* flag if the constraint set is unsatisfiable). For example, one mapping in this cache could be $(x + y < 10) \wedge (x > 5) \Rightarrow \{x = 6, y = 3\}$. Using these mappings, KLEE can quickly answer several types of similar queries, involving subsets and supersets of the constraint sets already cached. For example, if a subset of a cached constraint set is encountered, KLEE can simply return the cached solution, because removing constraints from a constraint set does not invalidate an existing solution. Moreover, if a superset of a cached constraint set is encountered, KLEE can quickly check if the cached solution still works, by plugging in those values into the superset. For example, KLEE can quickly check that $\{x = 6, y = 3\}$ is still a valid solution for the query $(x + y < 10) \wedge (x > 5) \wedge (y \geq 0)$, which is a superset of $(x + y < 10) \wedge (x > 5)$. This latter technique exploits the fact that in practice, adding extra constraints often does not invalidate an existing solution.

Memory Modeling. The precision with which program statements are translated into symbolic constraints can have a significant influence on the coverage achieved by symbolic execution, as well as on the scalability of constraint solving. For example, using a memory model that approximates fixed-width integer variables with actual mathematical integers may be more efficient, but on the other hand may result in imprecision in the analysis of code depending on corner cases such as arithmetic overflow—which may cause symbolic execution to miss paths, or explore infeasible ones.

Another example are pointers. On the one end of the spectrum is a system like DART that only reasons about concrete pointers, or systems like CUTE and CREST that support only equality and inequality constraints for pointers, which can be efficiently solved.³⁵ At the other end are systems like EXE, and more recently KLEE and SAGE^{10,17,35} that model pointers using the theory of

arrays with selections and updates implemented by solvers like STP or Z3.^{15,18}

The trade-off between precision and scalability should be determined in light of the code being analyzed (for example, low-level systems code vs. high-level applications code), and the exact performance difference between different constraint solving theories. Note that the trade-off between precision and scalability is possible in modern symbolic execution techniques because we can customize the use of concrete values in symbolic formulas and thereby tune both scalability and precision.

Handling Concurrency. Large real-world programs are often concurrent. Because of the inherent non-determinism of such programs, testing is notoriously difficult. Concolic testing was successfully combined with a variant of partial order reduction to test concurrent programs effectively.^{31–34} This combined method provides one of the first techniques to effectively test concurrent programs with complex data inputs.

Tools

Dynamic symbolic execution has been implemented by several tools from both academia and research labs.^{1,7–10,19,20,35,37} These tools support a variety of languages, including C/C++, Java, and the x86 instruction set, implement several different memory models, target different types of applications, and make use of several different constraint solvers and theories. We discuss here five of these tools, with whom the authors of this article have been involved.

DART, CUTE, and CREST. DART¹⁹ is the first concolic testing tool that combines dynamic test generation with random testing and model checking techniques with the goal of systematically executing all (or as many as possible) feasible paths of a program, while checking each execution for various types of errors. DART was first implemented at Bell Labs for testing C programs, and has inspired many other extensions and tools since.

CUTE (A Concolic Unit Testing Engine) and jCUTE (CUTE for Java)^{31,33,35} extend DART to handle multithreaded programs that manipulate dynamic data structures using pointer opera-

tions. In multithreaded programs, CUTE combines concolic execution with dynamic partial order reduction to systematically generate both test inputs and thread schedules.

CUTE and jCUTE were developed at University of Illinois at the Urbana-Champaign for C and Java programs, respectively. Both tools have been applied to several popular open source software including the java.util library of Sun JDK 1.4.

CREST⁷ is an open source tool for concolic testing of C programs. CREST is an extensible platform for building and experimenting with heuristics for selecting which paths to explore. Since being released as open source in May 2008,³ CREST has been downloaded 1,500+ times and has been used by several research groups. For example, CREST has been employed to build tools for augmenting existing test suites to test newly changed code³⁸ and detect SQL injection vulnerabilities,²⁹ has been modified to run distributed on a cluster for testing a flash storage platform,²² and has been used to experiment with more sophisticated concolic search heuristics.³

Concolic testing has also been studied in different courses at several universities.

EXE and KLEE. EXE¹⁰ is a symbolic execution tool for C designed for comprehensively testing complex software, with an emphasis on systems code. To deal with the complexities of systems code, EXE models memory with bit-level accuracy. This is needed because systems code often treats memory as untyped bytes, and observes a single memory location in multiple ways: for example, by casting signed variables to unsigned, or treating an array of bytes as a network packet, inode, or packet filter through pointer casting. As importantly, EXE provides the speed necessary to quickly solve the constraints generated by real code, through a combination of low-level optimizations implemented in its purposely designed constraint solver STP,^{10,18} and a series of higher-level ones such as caching and irrelevant constraint elimination.

KLEE⁸ is a redesign of EXE, built on top of the LLVM²⁴ compiler infra-

a Available at <http://code.google.com/p/crest/>

structure. Like EXE, it performs mixed concrete/symbolic execution, models memory with bit-level accuracy, employs a variety of constraint solving optimizations, and uses search heuristics to get high code coverage. One of the key improvements of KLEE over EXE is its ability to store a much larger number of concurrent states, by exploiting sharing among states at the object-, rather than at the page-level as in EXE. Another important improvement is its enhanced ability to handle interactions with the outside environment—for example, with data read from the file system or over the network—by providing models designed to explore all possible legal interactions with the outside world.

As a result of these features, EXE and KLEE have been successfully used to check a large number of different software systems, including network servers and tools (Berkeley Packet Filter, Avahi, Bonjour, among others);^{10,36} file systems (ext2, ext3, JFS);³⁹ MINIX device drivers (Sound Blaster 16, Lance, PCI);⁵ Unix utilities (Coreutils, MINIX, Busybox suites);⁸ and computer vision code.¹³ They exposed bugs and vulnerabilities in all of these software systems, and constructed concrete inputs triggering them. For example, EXE generated actual disk images that when mounted under various file systems cause the Linux kernel to panic.³⁹ EXE and KLEE were also able to successfully generate high-coverage regression suites: when run on the 89 stand-alone tools of the Coreutils utility suite, KLEE generates tests achieving on average over 90% line coverage, significantly beating an extensive manual regression suite built incrementally by developers over more than 15 years.

KLEE was open sourced in June 2009.^b The tool has an active user community—with approximately 200 members on the mailing list and growing—and several research groups have built upon it in a variety of areas, ranging from wireless sensor networks³⁰ to automated debugging,⁴⁰ reverse engineering of binary device drivers,¹⁴ exploit generation,² online gaming,⁴ testing and verification for GPUs,²⁵ and deterministic multithreading.¹⁴

^b Available at <http://klee.lvm.org/>

Conclusion

Symbolic execution has become an effective program testing technique, providing a way to automatically generate inputs that trigger software errors ranging from low-level program crashes to higher-level semantic properties; generate test suites that achieve high program coverage; and provide per-path correctness guarantees. While more research is needed to scale symbolic execution to very large programs, existing tools have already proved effective in testing and finding errors in a variety of software, varying from low-level network and operating systems code to higher-level applications code.

Acknowledgments

The EGT, EXE, and KLEE projects are joint work with Dawson Engler and several other researchers.^{5,8–10,13,36,39} Daniel Dunbar is the main author of the KLEE system. The DART and concolic testing projects are joint work with several researchers including Gul Agha, Jacob Burnim, Patrice Godefroid, Nils Klarlund, Rupak Majumdar, and Darko Marinov. C

References

- Anand, S., Păsăreanu, C.S. and Visser, W. JPF-SE: A symbolic execution extension to Java PathFinder. In *Proceedings of TACAS'07*.
- Avgerinos, T., Cha, S.K., Hao, B.L.T. and Brumley, D. AEG: Automatic exploit generation. In *Proceedings of NDSS'11*, (Feb. 2011).
- Baluda, M., Braione, P., Denaro, G. and Pezzè, M. Structural coverage of feasible code. In *Proceedings of AST'10*.
- Bethea, D., Cochran, R. and Reiter, M. Server-side verification of client behavior in online games. In *Proceedings of NDSS'10*, 2010.
- Boonstoppel, P., Cadar, C. and Engler, D. RWset: Attacking path explosion in constraint-based test generation. In *Proceedings of TACAS'08*, (Mar–Apr 2008).
- Boyer, R.S., Elspas, B. and Levitt, K.N. SELECT—A formal system for testing and debugging programs by symbolic execution. *SIGPLAN Not.* 10 (1975), 234–245.
- Burnim, J. and Sen, K. Heuristics for scalable dynamic test generation. In *Proceedings of ASE'08*, (Sept. 2008).
- Cadar, C., Dunbar, D. and Engler, D. KLEE: Unassisted and automatic generation of high-coverage tests for complex systems programs. In *Proceedings of OSDI'08*, (Dec 2008).
- Cadar, C. and Engler, D. Execution generated test cases: How to make systems code crash itself (invited paper). In *Proceedings of SPIN'05*, (Aug 2005).
- Cadar, C., Ganesh, V., Pawlowski, P., Dil, D. and Engler, D. EXE: Automatically generating inputs of death. In *Proceedings of CCS'06*, (Oct–Nov 2006). An extended version appeared in *ACM TISSEC* 12, 2 (2008).
- Chipounov, V. and Candea, G. Reverse engineering of binary device drivers with RevNIC. In *Proceedings of EuroSys'10*, (Apr 2010).
- Clarke, L.A. A program testing system. In *Proceedings of the 1976 Annual Conference*, 488–491.
- Collingbourne, P., Cadar, C. and Kelly, P.H. Symbolic crosschecking of floating-point and SIMD code. In *Proceedings of EuroSys'11*, (Apr 2011).
- Cui, H., Wu, J. che Tsai, C. and Yang, J. Stable deterministic multithreading through schedule memoization. In *Proceedings of OSDI'10*.

- De Moura, L. and Björner, N. Z3: An efficient SMT solver. In *Proceedings of TACAS'08*, (Mar–Apr 2008).
- De Moura, L. and Björner, N. Satisfiability modulo theories: introduction and applications. *Commun. ACM* 54, 9 (Sept. 2011), 69–77.
- Elkarablieh, B., Godefroid, P. and Levin, M.Y. Precise pointer reasoning for dynamic test generation. In *Proceedings of ISSTA'09*.
- Ganesh, V. and Dill, D.L. A decision procedure for bit-vectors and arrays. In *Proceedings of CAV'07*, (July 2007).
- Godefroid, P., Klarlund, N. and Sen, K. DART: Directed Automated Random Testing. In *Proceedings of PLDI'05*, (June 2005).
- Godefroid, P., Levin, M., and Molnar, D. Automated whitebox fuzz testing. In *Proceedings of NDSS'08*, (Feb. 2008).
- Hastings, R. and Joyce, B. Purify: Fast detection of memory leaks and access errors. In *Proceedings of Winter USENIX Conference*, 1992.
- Kim, Y., Kim, M., and Dang, N. Scalable distributed concolic testing: A case study on a flash storage platform. In *Proceedings of ICTAC'10*, 199–213.
- King, J.C. Symbolic execution and program testing. *Commun. ACM* 19, 7 (July 1976), 385–394.
- Lattner, C. and Adve, V. LLVM: A compilation framework for lifelong program analysis and transformation. In *Proceedings of CGO'04*, (Mar 2004).
- Li, G., Li, P., Sawaga, G., Gopalakrishnan, G., Ghosh, I. and Rajan, S.P. GKLEE: Concolic verification and test generation for GPUs. In *Proceedings of PPOPP'12*.
- Majumdar, R. and Sen, K. Hybrid concolic testing. In *Proceedings of ICSE'07*, (May 2007).
- Majumdar, R. and Sen, K. Latest: Lazy dynamic test input generation. Technical Report UCB/ECS-2007-36. EECs Department, University of California, Berkeley, Mar. 2007.
- Nethercote, N. and Seward, J. Valgrind: A program supervision framework. *Electronic Notes in Theoretical Computer Science* 89, 2 (2003).
- Ruse, M., Sarkar, T. and Basu, S. Analysis & detection of SQL injection vulnerabilities via automatic test case generation of programs. In *Proceedings of SAINT'10*, (July 2010).
- Sasnauskas, R., Link, J.A.B., Alizai, M.H., and Wehrle, K. Kleenet: Automatic bug hunting in sensor network applications. In *Proceedings of IPSN'10*, (Apr 2010).
- Sen, K. Scalable Automated Methods for Dynamic Program Analysis. Ph.D. thesis. University of Illinois at Urbana-Champaign, June 2006.
- Sen, K. and Agha, G. Automated systematic testing of open distributed programs. In *Proceedings of FASE'06*, 2006.
- Sen, K. and Agha, G. CUTE and jCUTE: Concolic unit testing and explicit path model-checking tools. In *Proceedings of CAV'06*.
- Sen, K. and Agha, G. A race-detection and flipping algorithm for automated testing of multi-threaded programs. In *Proceedings of HVC*, (2006).
- Sen, K., Marinov, D. and Agha, G. CUTE: A concolic unit testing engine for C. In *Proceedings of ESEC/FSE'05*, (Sept. 2005).
- Song, J., Ma, T., Cadar, C. and Pietzuch, P. Rule-based verification of network protocol implementations using symbolic execution. In *Proceedings of ICCCN'11*, (May 2011).
- Tillmann, N. and de Halleux, J. Pex—White box test generation for .NET. In *Proceedings of TAP'08*, (Apr. 2008).
- Xu, Z., Kim, Y., Kim, M., Rothermel, G. and Cohen, M.B. Directed test suite augmentation: Techniques and trade-offs. In *Proceedings of FSE'10*, (Nov. 2010).
- Yang, J., Sar, C., Twohey, P., Cadar, C. and Engler, D. Automatically generating malicious disks using symbolic execution. In *IEEE Symposium on Security and Privacy*, (May 2006).
- Zamfir, C. and Candea, G. Execution synthesis: A technique for automated software debugging. In *Proceedings of EuroSys'10*, (Apr 2010).

Cristian Cadar (c.cadar@imperial.ac.uk) is a lecturer in the Department of Computing at Imperial College London.

Koushik Sen (ksen@cs.berkeley.edu) is an associate professor in the Department of Electrical Engineering and Computer Science at the University of California, Berkeley.

research highlights

P. 92

Technical Perspective Is Dark Silicon Real?

By Pradip Bose

P. 93

Power Challenges May End the Multicore Era

By Hadi Esmaeilzadeh, Emily Blem, Renée St. Amant,
Karthikeyan Sankaralingam, and Doug Burger

Technical Perspective Is Dark Silicon Real?

By Pradip Bose

THE MICROPROCESSOR CHIP R&D community has been well aware of the so-called “power wall” challenge for over a decade now. Researchers have focused mainly on creative techniques to improve power-performance efficiency. Developers have adopted many of those ideas, and through engineering innovations have been able to keep the economics of technology scaling largely justifiable up until now. Industry witnessed a clear paradigm shift (as a response to the looming power wall) when the single-core processor chip era gave way to the multicore era at the beginning of the current century. Power (and power *density*) limits, coupled with the steady demise of idealized Dennard scaling rules, made it difficult to keep increasing the clock frequency. Also, limits in instruction-level parallelism (ILP) made it difficult to keep increasing single-core instructions-per-cycle (IPC), without spending an inordinate amount of area and power. However, even though we embarked upon the multicore trail, the power wall was never forgotten. We knew that sacrificing single-thread performance in favor of generational increases in chip throughput would not make the power wall go away forever! It would loom large again as the core count kept increasing. Because, fundamentally, the memory and I/O bandwidth demands dictated by the need to “feed” so many cores costs power and pins that we do not have. Also, delivering the current to feed an increasing number of cores at a lower voltage than before makes the designer hit a chip C4 current limit wall that is difficult to ignore.


The following work by Esmaeilzadeh et al. is a landmark paper that opens our eyes to the unrelenting power challenge we face in the multicore era. Most interestingly, it raises the specter of *dark silicon*: lots of processor cores, but very few that can be powered on or utilized at any given time. Not that the authors see this as a desirable feature of future designs;

but they certainly raise a very valid question about the future viability of the basic multicore paradigm. Just like ILP limits make it ever harder to boost single-thread IPC at affordable power and complexity, thread-level parallelism (TLP) at the chip level gets ever harder because of the limited parallelism in so-called parallel applications. And, even for some scientific applications that are embarrassingly parallel or for commercial server workloads with large TLP, on-chip shared hardware resource contention and size limits make it more difficult to extract that parallelism at affordable power. So, even if we are able to go on doubling the number of cores each technology generation, we have two basic problems, as clearly enunciated by the authors: for a fixed chip power budget and area, even a very aggressive investment in application parallelism enhancement does not help one get even respectably close to the targeted 2X (throughput) performance growth per generation; and even if cooling and power delivery technologies improve to allow a large increase in the chip power budget, real application parallelism levels would not allow targeted performance scaling in most cases—not by a long shot. The paper’s elegant analytical formalism shows that under ITRS projections, as we approach the 8nm technology node, over half the chip will remain unutilized (and consequently “dark”). In a sense, this is regardless of whether one views the problem from the perspective of a power wall constraint or from one that focuses first on the effective TLP limit constraint.

The effective parallelism content of real application workloads is often small enough that strong single-thread performance remains a crucial factor to combat the (serial) Amdahl bottleneck. The paper, therefore does consider heterogeneous (or asymmetric) multicores in the analysis in a quest to find an optimistic outlook for

the future. However, the combination of realistic chip power limits and real application parallelism limits makes it hard or impossible to sustain historically established performance growth rates using the multicore paradigm as we currently know it.

Is the specter of progressively darker silicon real? Or, are there technological or design breakthroughs around the corner to help us circumvent such a scenario, at least in the short term? Alternatively, if that specter is indeed real from a utilization efficiency viewpoint, but not directly from a power limit perspective, are there other ways the “idle” cores can be used to provide functionality that is not traditional “performance”? For example, can available idle cores be used to enhance reliability or security? The paper does briefly journey into optimistic dreamland to give the reader a hint about promising new innovations that could potentially be disruptive in the face of the specter that seems to be haunting us at this time.

This paper is not just a doomsday predictor. It raises our awareness of the problem through scientific quantification; but it should also serve as a springboard for innovative research, especially for computer architects. However, the architect cannot hope to invent in a vacuum; the needed innovations will surely come, but only by adopting a holistic, cross-layer view of the full system—from devices, through circuits, microarchitecture, system architecture, and the software stack. Researchers are well-aware of this urgent need, thanks to papers like this one; the industrial development teams cannot wait to take advantage of the next generation of holistic, cross-layer system architectural thoughts, models, and design tools. 

Pradip Bose (pbose@us.ibm.com) is a research staff member at IBM T.J. Watson Research Center where he manages the Reliability- and Power-Aware Microarchitectures department.

© 2013 ACM 0001-0782/13/02

Power Challenges May End the Multicore Era

By Hadi Esmaeilzadeh, Emily Blem, Renée St. Amant, Karthikeyan Sankaralingam, and Doug Burger

Abstract

Starting in 2004, the microprocessor industry has shifted to multicore scaling—increasing the number of cores per die each generation—as its principal strategy for continuing performance growth. Many in the research community believe that this exponential core scaling will continue into the hundreds or thousands of cores per chip, auguring a parallelism revolution in hardware or software. However, while transistor count increases continue at traditional Moore's Law rates, the per-transistor speed and energy efficiency improvements have slowed dramatically. Under these conditions, more cores are only possible if the cores are slower, simpler, or less utilized with each additional technology generation. This paper brings together transistor technology, processor core, and application models to understand whether multicore scaling can sustain the historical exponential performance growth in this energy-limited era. As the number of cores increases, power constraints may prevent powering of all cores at their full speed, requiring a fraction of the cores to be powered off at all times. According to our models, the fraction of these chips that is "dark" may be as much as 50% within three process generations. The low utility of this "dark silicon" may prevent both scaling to higher core counts and ultimately the economic viability of continued silicon scaling. Our results show that core count scaling provides much less performance gain than conventional wisdom suggests. Under (highly) optimistic scaling assumptions—for parallel workloads—multicore scaling provides a 7.9× (23% per year) over ten years. Under more conservative (realistic) assumptions, multicore scaling provides a total performance gain of 3.7× (14% per year) over ten years, and obviously less when sufficiently parallel workloads are unavailable. Without a breakthrough in process technology or microarchitecture, other directions are needed to continue the historical rate of performance improvement.

1. INTRODUCTION

Moore's Law¹⁸ (the doubling of transistors on chip every 18 months) has been a fundamental driver of computing. For more than four decades, through transistor, circuit, microarchitecture, architecture, and compiler advances, Moore's Law, coupled with Dennard scaling,⁹ has resulted in consistent exponential performance increases. Dennard's scaling theory showed how to reduce the dimensions and the electrical characteristics of a transistor proportionally to enable successive shrinks that simultaneously improved density, speed, and energy efficiency. According to Dennard's theory with a scaling ratio of $\frac{1}{\sqrt{2}}$, the transistor count doubles (Moore's Law), frequency increases by 40%, and the total

chip power stays the same from one generation of process technology to the next on a fixed chip area. With the end of Dennard scaling, process technology scaling can sustain doubling the transistor count every generation, but with significantly less improvement in transistor switching speed and energy efficiency. This transistor scaling trend presages a divergence between energy efficiency gains and transistor density increases. The recent shift to multicore designs, which was partly a response to the end of Dennard scaling, aimed to continue proportional performance scaling by utilizing the increasing transistor count to integrate more cores, which leverage application and/or task parallelism.

Given the transistor scaling trends and challenges, it is timely and crucial for the broader computing community to examine whether multicore scaling will utilize each generation's doubling transistor count effectively to sustain the performance improvements we have come to expect from technology scaling. Even though power and energy have become the primary concern in system design, no one knows how severe (or not) the power problem will be for multicore scaling, especially given the large multicore design space.

Through comprehensive modeling, this paper provides a decade-long performance scaling projection for future multicore designs. Our multicore modeling takes into account transistor scaling trends, processor core design options, chip multiprocessor organizations, and benchmark characteristics, while applying area and power constraints at future technology nodes. The model combines these factors to project the upper bound speedup achievable through multicore scaling under current technology scaling trends. The model also estimates the effects of nonideal transistor scaling, including the percentage of *dark silicon*—the fraction of the chip that needs to be powered off at all times—in future multicore chips. Our modeling also discovers the best core organization, the best chip-level topology, and the optimal number of cores for the workloads studied. We do not believe or advocate that designs with dark silicon are ideal or even desirable; in our view smaller chips are more likely. Nonetheless, our modeling shows that—even with the best multicore organization, assuming constant chip size and fixed power budget—a significant portion of the chip will remain dark.

The study shows that regardless of chip organization and topology, multicore scaling is power limited to a degree not

A previous version of this article appears in *Proceedings of the 38th International Symposium on Computer Architecture* (June 2011). Parts of this article appear in *IEEE Micro Top Picks from the Computer Architecture Conferences of 2011* (May/June 2012).

widely appreciated by the computing community. In just five generations, at 8nm, the percentage of dark silicon in a fixed-size chip may grow to 50%. Given the recent trend of technology scaling, the 8nm technology node is expected to be available in 2018. Over this period of ten years (from 2008 when 45nm microprocessors were available), with optimistic international technology roadmap for semiconductors (ITRS) scaling projections,¹⁶ only 7.9× average speedup is possible for commonly used parallel workloads,⁴ leaving a nearly 24-fold gap from a target of doubled performance per generation. This gap grows to 28-fold with conservative scaling projections,⁵ with which only 3.7× speedup is achievable in the same period. Further investigations also show that beyond a certain point increasing the core count does not translate to meaningful performance gains. These power and parallelism challenges threaten to end the multicore era, defined as the era during which core counts grow appreciably.

2. OVERVIEW

Figure 1 shows how we build and combine three models to project the performance of future multicores. Ultimately, the model predicts the speedup achievable by multicore scaling and shows a gap between our model’s projected speedup and the expected exponential speedup with each technology generation. We refer to this gap as the *dark silicon performance gap*, since it is partly the result of the dark silicon phenomenon, or the nonideal transistor scaling that prevents fully utilizing the exponential increases in transistor count. Our modeling considers transistor scaling projections, single-core design scaling, multicore design choices, application characteristics, and microarchitectural features. This study assumes that the die size and the power budget stay the same as technology scales,

an assumption in line with the common practice for microprocessor design. Below we briefly discuss each of the three models.

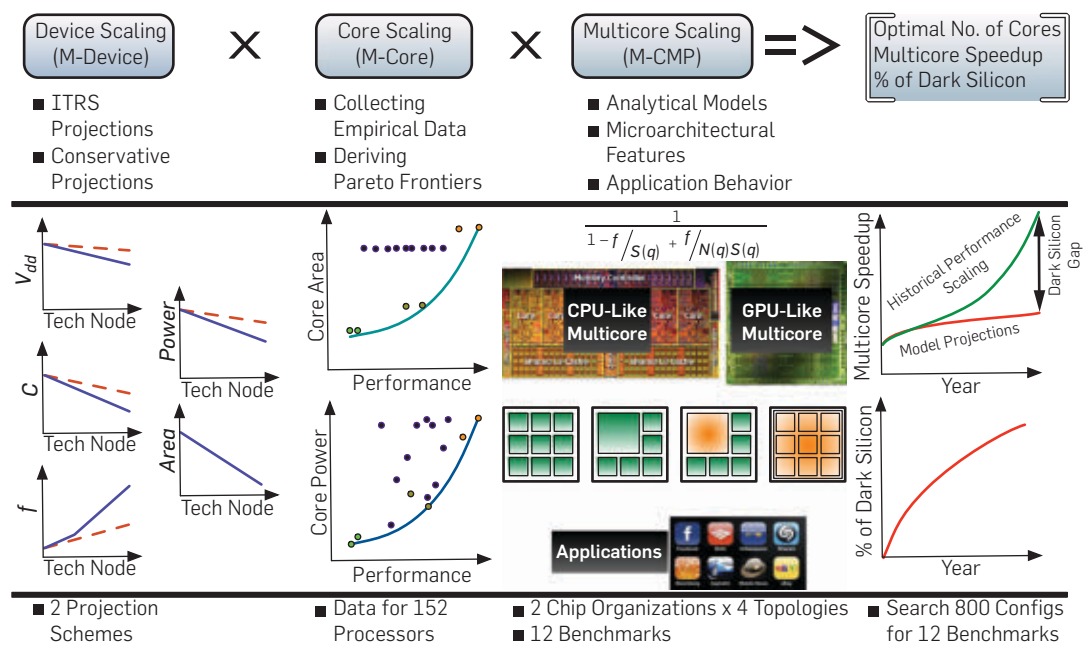
Device scaling model (M-Device). Two device (transistor) scaling models provide the area, power, and frequency scaling factors at technology nodes from 45nm through 8nm. One model is based on aggressive ITRS projections¹⁶ while the other builds on more conservative predictions from Shekhar Borkar’s recent study.⁵

Core scaling model (M-Core). Through Pareto-optimal curves, derived from measured data, the M-Core model provides the maximum performance that a single-core can sustain for any given area. Further, it provides the minimum power that is consumed to sustain this level of performance. At each technology node, these two Pareto frontiers, which constitute the M-Core model, define the best-case design space of single cores.

Multicore scaling model (M-CMP). The M-CMP covers two mainstream classes of multicore organizations, multicore CPUs and many-thread GPUs, which represent two extreme points in the threads-per-core spectrum. The CPU multicore organization represents Intel Nehalem-like multicore designs that benefit from large caches and offer relatively high single-thread performance. The GPU multicore organization represents NVIDIA Tesla-like lightweight cores with heavy multithreading support and poor single-thread performance. In modeling each of the two multicore organizations, we consider four topologies: symmetric, asymmetric, dynamic, and composed (also called “fused” in the literature¹⁵).

Symmetric multicore. The symmetric, or homogeneous, multicore topology consists of multiple copies of the same core operating at the same voltage and frequency

Figure 1. Overview of the methodology and the models.



setting. In a symmetric multicore, the resources, including the power and the area budget, are shared equally across all cores.

Asymmetric multicore. The asymmetric multicore topology consists of one large monolithic core and many identical small cores. This design uses the high-performing large core for the serial portion of code and leverages the numerous small cores as well as the large core to exploit the parallel portion of code.

Dynamic multicore. The dynamic multicore topology is a variation of the asymmetric multicore topology. During parallel code portions, the large core is shut down and, conversely, during the serial portion, the small cores are turned off and the code runs only on the large core.^{6, 21} Switching the cores off and on allows integrating more cores or using a higher voltage and frequency operational setting.

Composed multicore. The composed multicore topology is a collection of small cores that can dynamically merge together and compose a logical higher performance large core.^{15, 17} While providing a parallel substrate for the parallel portion of code when unmerged, the small cores merge and compose a logical core that offers higher single-threaded performance for the serial portion.

The multicore model is an analytic model that computes the multicore performance and takes the core performance as input (obtained from M-Core), the multicore organization (CPU-like or GPU-like), and multicore topology (symmetric, asymmetric, dynamic, and composed). Unlike previous studies, the model also takes into account application characteristics such as memory access pattern, and the amount of thread-level parallelism in the workload as well as the microarchitectural features such as cache size and memory bandwidth. We choose the PARSEC benchmarks⁴ to study the multicore scaling potential for successfully parallelized applications. PARSEC is a set of highly parallel applications that are widely used to support the parallel architecture research.

Modeling future multicore chips. To model future multicore chips, we first model the building blocks, the future cores. We combine the device and core models to project the best-case design space of single cores—the Pareto frontiers—at future technology nodes. Any performance improvement for future cores will come at the cost of area or power as defined by the projected Pareto frontiers. Then, we combine all three models and perform an exhaustive design-space search to find the optimal multicore configuration for each individual application considering its characteristics. The optimal configuration delivers the maximum multicore speedup for each benchmark at future technology nodes while enforcing area and power constraints. The gap between the projected speedup and the speedup we have come to expect with each technology generation is the dark silicon performance gap.

Related work. Other work has studied various subsets of the problem that we study comprehensively. Hill and Marty extend Amdahl's Law to model multicore architectures with different topologies.¹⁴ Hempstead et al. introduce a variant of Amdahl's Law to estimate the amount of specialization required to maintain $1.5\times$ performance growth per year,

assuming completely parallelizable code.¹³ Chung et al. study unconventional cores including custom logic, FPGAs, or GPUs in heterogeneous single-chip designs.⁷ Azizi et al. derive the single-core energy/performance tradeoff as Pareto frontiers using architecture-level statistical models combined with circuit-level energy-performance tradeoff functions.² Chakraborty considers device-scaling and estimates a simultaneous activity factor for technology nodes down to 32nm.⁶ Venkatesh et al. estimate technology-imposed utilization limits and motivate energy-efficient and application-specific core designs.²² Hardavellas et al. forecast the limits of multicore scaling and the emergence of dark silicon in servers with workloads that have an inherent abundance of parallelism.¹²

3. DEVICE MODEL (M-DEVICE)

The first step in projecting gains from more cores is developing a model that captures future transistor scaling trends. To highlight the challenges of nonideal device scaling, first we present a simplified overview of historical Dennard scaling and the more recent scaling trends.

Historical device scaling trends. According to Dennard scaling, as the geometric dimensions of transistors scale, the electric field within the transistors stays constant if other physical features, such as the gate oxide thickness and doping concentrations, are reduced proportionally. To keep the electric field constant, the supply voltage (the switch on voltage) as well as the threshold voltage (the voltage level below which the transistor switches off) need to be scaled at the same rate as the dimensions of the transistor. With Dennard scaling, a 30% reduction in transistor length and width results in a 50% decrease in transistor area, doubling the number of transistors that can fit on chip with each technology generation (Moore's Law¹⁸). Furthermore, the decrease in transistor sizes results in a 30% reduction in delay. In total, Dennard scaling suggests a 30% reduction in delay (hence 40% increase in frequency), a 50% reduction in area, and a 50% reduction in power per transistor. As a result, the chip power stays the same as the number of transistors doubles from one technology node to the next in the same area.

Recent device scaling trends. At recent technology nodes, the rate of supply voltage scaling has dramatically slowed due to limits in threshold voltage scaling. Leakage current increases exponentially when the threshold voltage is decreased, limiting threshold voltage scaling, and making leakage power a significant and first-order constraint. Additionally, as technology scales to smaller nodes, physics limits decrease in gate oxide thickness. These two phenomena were not considered in the original Dennard scaling theory, since leakage power was not dominant in the older generations, and the physical limits of scaling oxide thickness were too far out to be considered. Consequently, Dennard scaling stopped at 90nm.⁸ That is, transistor area continues to scale at the historic rate, which allows for doubling the number of transistors, while the power per transistor is not scaling at the same rate. This disparity will translate to an increase in chip power if the fraction of active transistors is not reduced from one technology generation

to the next.⁶ The shift to multicore architectures was partly a response to the end of Dennard scaling.

3.1. Model structure

The device model provides transistor area, power, and frequency scaling factors from a base technology node (e.g. 45nm) to future technologies. The area scaling factor corresponds to the shrinkage in transistor dimensions. The frequency scaling factor is calculated based on the fan-out of 4 (FO4) delay reduction. FO4 is a process independent delay metric used to measure the delay of CMOS logic that identifies the processor frequency. FO4 is the delay of an inverter, driven by an inverter 4× smaller than itself, and driving an inverter 4× larger than itself. The power scaling factor is computed using the predicted frequency, voltage, and gate capacitance scaling factors in accordance with the $P = \alpha CV_{dd}^2 f$ equation.

3.2. Model implementation

We generate two device scaling models: ITRS and conservative. The ITRS model uses projections from the ITRS 2010 technology roadmap.¹⁶ The conservative model is based on predictions by Shekhar Borkar and represents a less optimistic view.⁵ The parameters used for calculating the power and performance scaling factors are summarized in Table 1. We allocate 20% of the chip-power budget to leakage power and assume chip designers can maintain this ratio.

4. CORE MODEL (M-CORE)

The second step in estimating future multicore performance is modeling a key building block, the processor core.

4.1. Model structure

We build a technology-scalable core model by populating the area/performance and power/performance design spaces with the data collected for a set of processors; all fabricated in the same technology node. The core model is the combination of the area/performance Pareto frontier, $A(q)$, and the

Table 1. Scaling factors with ITRS and conservative projections.

	Tech node (nm)	Frequency scaling factor (/45nm)	V_{dd} scaling factor (/45nm)	Capacitance scaling factor (/45nm)	Power scaling factor (/45nm)
ITRS	45*	1.00	1.00	1.00	1.00
	32*	1.09	0.93	0.70	0.66
	22 [†]	2.38	0.84	0.33	0.54
	16 [†]	3.21	0.75	0.21	0.38
	11 [†]	4.17	0.68	0.13	0.25
	8 [†]	3.85	0.62	0.08	0.12
	31% frequency increase and 35% power reduction per node				
Conservative	45	1.00	1.00	1.00	1.00
	32	1.10	0.93	0.75	0.71
	22	1.19	0.88	0.56	0.52
	16	1.25	0.86	0.42	0.39
	11	1.30	0.84	0.32	0.29
	8	1.34	0.84	0.24	0.22
6% frequency increase and 23% power reduction per node					

*Extended planar bulk transistors.

[†]Multi-gate transistors.

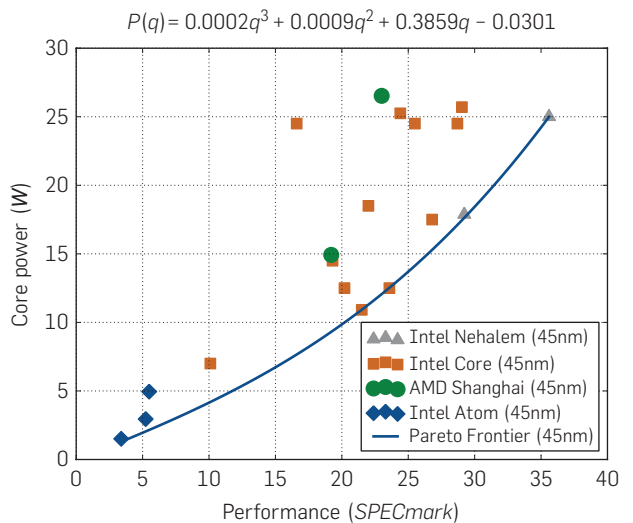
power/performance Pareto frontier, $P(q)$, for these two design spaces, where q is the single-threaded performance of a core. These frontiers capture the best-case area/performance and power/performance tradeoffs for a core while abstracting away specific details of the core. We use the device scaling model to project the frontiers to future technologies and model performance, area, and power of cores fabricated at those nodes.

4.2. Model implementation

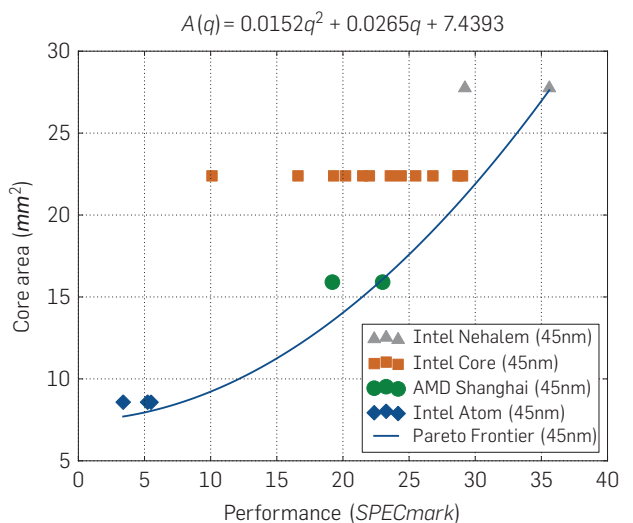
As Figure 2 depicts, we populate the two design spaces at 45nm using 20 representative Intel and AMD processors and derive the Pareto frontiers. The curve that bounds all power(area)/performance points in the design space and minimizes power(area) required for a given level of performance constructs the Pareto frontier. The polynomials $P(q)$ and $A(q)$ are the core model. The core performance, q , is the processor's SPECmark and is collected from the SPEC

Figure 2. Design space and derivation of the Pareto frontiers.

(a) Power/performance frontier, 45nm



(b) Area/performance frontier, 45nm



Website.²⁰ We estimate the core power budget using the TDP reported in processor datasheets. TDP is the chip-power budget, or the amount of power the chip can dissipate without exceeding the transistor junction temperature. We use die photos of the four microarchitectures, Intel Atom, Intel Core, AMD Shanghai, and Intel Nehalem, to estimate the core areas (excluding level 2 and level 3 caches). Since the focus of this work is to study the impact of technology constraints on logic scaling rather than cache scaling, we derive the Pareto frontiers using only the portion of power budget and area allocated to the core in each processor excluding the uncore components.

As illustrated in Figure 2, a cubic polynomial, $P(q)$, is fit to the points along the edge of the power/performance design space and a quadratic polynomial (Pollack's rule¹⁹), $A(q)$, to the points along the edge of the area/performance design space. The Intel Atom Z520 with an estimated 1.89W TDP per core represents the lowest power design (lower-left frontier point), and the Nehalem-based Intel Core i7-965 Extreme Edition with an estimated 31.25W TDP per core represents the highest performing design (upper-right frontier point). The points along the scaled Pareto frontier are used as the search space for determining the best core configuration by the multicore model.

5. MULTICORE MODEL (M-CMP)

The last step in modeling multicore scaling is to develop a detailed chip-level model (M-CMP) that integrates the area and power frontiers, microarchitectural features, and application behavior, while accounting for the chip organization (CPU-like or GPU-like) and its topology (symmetric, asymmetric, dynamic, or composed).

5.1. Model structure

Guz et al. proposed a model to consider first-order impacts of microarchitectural features (cache organization, memory bandwidth, number of threads per core, etc.) and workload characteristics (memory access pattern).¹⁰ To first order, their model considers stalls due to memory dependences and resource constraints (bandwidth or functional units). We extend their approach to build our multicore model. Our extensions incorporate additional application characteristics, microarchitectural features, and physical constraints, and covers both homogeneous and heterogeneous multicore topologies.

This model uses single-threaded cores with large caches to cover the CPU multicore design space and massively threaded cores with minimal caches to cover the GPU multicore design while modeling all four topologies. The input parameters to the model, and how, if at all, they are affected by the multicore design choices are listed in Table 2.

Multicore topologies. The multicore model is an extended Amdahl's Law¹ equation that incorporates the multicore performance ($Perf$) calculated from (2)–(5):

$$Speedup = 1 / \left(\frac{f}{S_{Parallel}} + \frac{1-f}{S_{Serial}} \right) \quad (1)$$

The M-CMP model (1) measures the multicore speedup with respect to a baseline multicore ($Perf_b$). That is, the

Table 2. M-CMP parameters with default values from 45nm Nehalem.

Parameter	Description	Default	Affected by
N	Number of cores	4	Multicore topology
T	Number of threads per core	1	Core style
$freq$	Core frequency (MHz)	3200	Core performance
CPI	Cycles per instruction (zero-latency cache accesses)	1	Core performance, application
C_1	L1 cache size per core (KB)	64	Core style
C_2	L2 cache size per chip (MB)	2	Core style, multicore topology
t_1	L1 access time (cycles)	3	–
t_2	L2 access time (cycles)	20	–
t	Memory access time (cycles)	426	Core performance
BW	Maximum memory bandwidth (GB/s)	200	Technology node
b	Bytes per memory access (B)	64	–
f	Fraction of code that can be parallel	Varies	Application
r	Fraction of instructions that are memory accesses	Varies	Application
α_1, β_1	L1 cache miss rate function constants	Varies	Application
α_2, β_2	L2 cache miss rate function constants	Varies	Application

parallel portion of code (f) is sped up by $S_{Parallel} = Perf_p / Perf_b$ and the serial portion of code ($1 - f$) is sped up by $S_{Serial} = Perf_s / Perf_b$.

The number of cores that fit on the chip is calculated as follows based on the topology of the multicore, its area budget ($AREA$), its power budget (TDP), each core's area ($A(q)$), and each core's power ($P(q)$).

$$N_{symm}(q) = \min \left(\frac{AREA}{A(q)}, \frac{TDP}{P(q)} \right)$$

$$N_{Asym}(q_L, q_S) = \min \left(\frac{AREA - A(q_L)}{A(q_S)}, \frac{TDP - P(q_L)}{P(q_S)} \right)$$

$$N_{dynam}(q_L, q_S) = \min \left(\frac{AREA - A(q_L)}{A(q_S)}, \frac{TDP}{P(q_S)} \right)$$

$$N_{comp}(q_L, q_S) = \min \left(\frac{AREA}{(1 + \tau)A(q_S)}, \frac{TDP}{P(q_S)} \right)$$

For heterogeneous multicores, q_S is the single-threaded performance of the small cores and q_L is the single-threaded performance of the large core. The area overhead of supporting composability is τ ; however, no power overhead is assumed for composability support.

Microarchitectural features. Multithreaded performance ($Perf$) of an either CPU-like or GPU-like multicore running a fully parallel ($f = 1$) and multithreaded application is

calculated in terms of instructions per second in (2) by multiplying the number of cores (N) by the core utilization (η) and scaling by the ratio of the processor frequency to CPI_{exe} :

$$Perf = \min \left(N \cdot \frac{freq}{CPI_{exe}} \eta, \frac{BW_{max}}{r_m \times m_{L1} \times m_{L2} \times b} \right) \quad (2)$$

The CPI_{exe} parameter does not include stalls due to cache accesses, which are considered separately in the core utilization (η). The core utilization is the fraction of time that a thread running on the core can keep it busy. It is modeled as a function of the average time spent waiting for each memory access (t), fraction of instructions that access the memory (r_m), and the CPI_{exe} :

$$\eta = \min \left(1, \frac{T}{1 + t \frac{r_m}{CPI_{exe}}} \right) \quad (3)$$

The average time spent waiting for memory accesses (t) is a function of the time to access the caches (t_{L1} and t_{L2}), time to visit memory (t_{mem}), and the predicted cache miss rate (m_{L1} and m_{L2}):

$$t = (1 - m_{L1})t_{L1} + m_{L1}(1 - m_{L2})t_{L2} + m_{L1}m_{L2}t_{mem} \quad (4)$$

$$m_{L1} = \left(\frac{C_{L1}}{T\beta_{L1}} \right)^{1-\alpha_{L1}} \quad \text{and} \quad m_{L2} = \left(\frac{C_{L2}}{NT\beta_{L2}} \right)^{1-\alpha_{L2}} \quad (5)$$

5.2. Model implementation

The M-CMP model incorporates the Pareto frontiers, physical constraints, real application characteristics, and realistic microarchitectural features into the multicore speedup projections as discussed below.

Application characteristics. The input parameters that characterize an application are its cache behavior, fraction of instructions that are loads or stores, and fraction of parallel code. For the PARSEC benchmarks, we obtain this data from two previous studies.^{3,4} To obtain the fraction of parallel code (f) for each benchmark, we fit an Amdahl's Law-based curve to the reported speedups across different numbers of cores from both studies. The value of f ranges from 0.75 to 0.9999 depending on the benchmark.

Obtaining frequency and CPI_{exe} from Pareto frontiers. To incorporate the Pareto-optimal curves into the M-CMP model, we convert the SPECmark scores (q) into an estimated CPI_{exe} and core frequency. We assume the core frequency scales linearly with performance, from 1.5 GHz for an Atom core to 3.2 GHz for a Nehalem core. Each application's CPI_{exe} is dependent on its instruction mix and use of hardware resources (e.g., functional units and out-of-order issue width). Since the measured CPI_{exe} for each benchmark at each technology node is not available, we use the M-CMP model to generate per benchmark CPI_{exe} estimates for each design point along the Pareto frontier. With all other model inputs kept constant, we iteratively search for the CPI_{exe} at

each processor design point. We start by assuming that the Nehalem core has a CPI_{exe} of ℓ . Then, the smallest core, an Atom processor, should have a CPI_{exe} such that the ratio of its M-CMP performance to the Nehalem core's M-CMP performance is the same as the ratio of their SPECmark scores (q). We assume CPI_{exe} does not change as technology scales, while frequency does change as discussed in Section 6.1.

Microarchitectural features. A key part of the detailed model is the set of input parameters that model the microarchitecture of the cores. For single-thread (ST) cores, we assume each core has a 64KB L1 cache, and chips with only ST cores have an L2 cache that is 30% of the chip area. Many-thread (MT) cores have small L1 caches (32KB for every 8 cores), support multiple hardware contexts (1024 threads per 8 cores) and a thread register file, and have no L2 cache. From Atom and Tesla die photos, we estimate that 8 small MT cores, their shared L1 cache, and their thread register file can fit in the same area as one Atom processor. We assume that off-chip bandwidth (BW_{max}) increases linearly as process technology scales down while the memory access time is constant.

Composed multicores. We assume that τ (area overhead of composability) increases from 10% to 400% depending on the total area of the composed core and performance of the composed core cannot exceed performance of a single Nehalem core at 45nm.

Constraints and baseline. The area and power budgets are derived from the highest-performing quad-core Nehalem multicore at 45nm excluding the L2 and L3 caches. They are 111 mm^2 and 125 W , respectively. The M-CMP multicore speedup baseline is a quad-Nehalem multicore that fits in the area and power budgets. The reported dark silicon projections are for the area budget that is solely allocated to the cores, not caches and other 'uncore' components. The actual fraction of chip that goes dark may be higher.

6. COMBINING MODELS

6.1. Device \times core model

To study core scaling in future technology nodes, we scaled the 45nm Pareto frontiers down to 8nm by scaling the power and performance of each processor data point using the *DevM* model and then re-fitting the Pareto optimal curves at each technology node. Performance, measured in SPECmark, is assumed to scale linearly with frequency. This optimistic assumption ignores the effects of memory latency and bandwidth on the core performance, and thus actual performance gains through scaling may be lower. Based on the optimistic ITRS model, scaling a microarchitecture (core) from 45nm to 8nm will result in a 3.9 \times performance improvement and an 88% reduction in power consumption. Conservative scaling, however, suggests that performance will increase only by 34%, and power will decrease by 74%.

6.2. Device \times core \times multicore model

All three models are combined to produce final projections on optimal multicore speedup, optimal number of cores, and amount of dark silicon. To determine the

best multicore configuration at each technology node, we sweep the design points along the scaled area/performance and power/performance Pareto frontiers (M-Device \times M-Core) as these points represent the most efficient designs. At each technology node, for each core design on the scaled frontiers, we construct a multicore chip consisting of one such core. For a symmetric multicore chip, we iteratively add identical cores one by one until the area or power budget is hit, or performance improvement is limited. We sweep the frontier and construct a symmetric multicore for each processor design point. From this set of symmetric multicores, we pick the multicore with the best speedup as the optimal symmetric multicore for that technology node. The procedure is similar for other topologies. This procedure is performed separately for CPU-like and GPU-like organizations. The amount of dark silicon is the difference between the area occupied by cores for the optimal multicore and the area budget allocated to the cores.

7. SCALING AND FUTURE MULTICORES

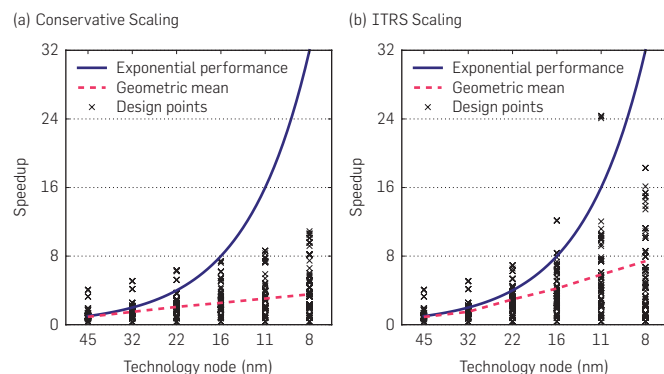
We apply the combined models to study the future of multicore designs and their performance-limiting factors. The results from this study provide a detailed analysis of multicore behavior for future technologies considering 12 real applications from the PARSEC suite.

7.1. Speedup projections

Figure 3 summarizes all of the speedup projections in a single scatter plot. For every benchmark at each technology node, we plot the speedup of eight possible multicore configurations (CPU-like, GPU-like) \times (symmetric, asymmetric, dynamic, composed). The solid line is exponential performance scaling—doubling performance every technology generation.

With optimal multicore configurations for each individual application, at 8nm, only 3.7 \times (conservative scaling) or 7.9 \times (ITRS scaling) geometric mean speedup is possible, as shown by the dashed line in Figure 3.

Figure 3. Speedup across process technology nodes across all organizations and topologies with PARSEC benchmarks.



Highly parallel workloads with a degree of parallelism higher than 99% will continue to benefit from multicore scaling.

At 8nm, the geometric mean speedup for dynamic and composed topologies is only 10% higher than the geometric mean speedup for symmetric topologies.

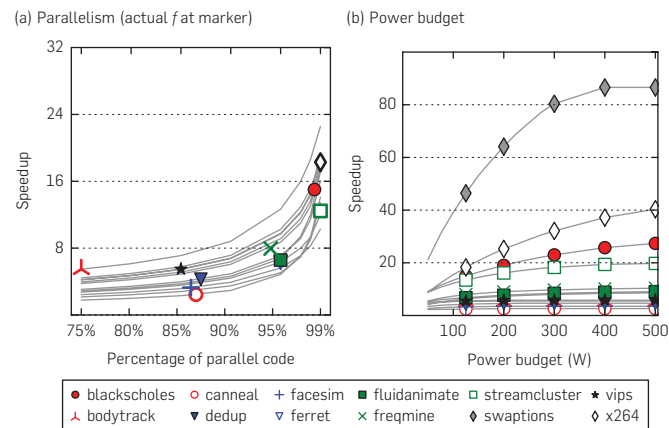
7.2. Dark silicon projections

With ITRS projections, at 8nm, over 50% of the chip will be dark and cannot be utilized.

To understand whether parallelism or the power constraint is the primary source of the dark silicon performance gap, we vary each of these factors in two experiments at 8nm. First, as depicted in Figure 4(a), we keep the power budget constant (our default budget is 125 W), and vary the level of parallelism in the PARSEC applications from 0.75 to 0.99, assuming programmer effort can realize this improvement. We see performance improves slowly as the parallelism level increases, with most benchmarks reaching a speedup of about only 15 \times at 99% parallelism. Provided that the power budget is the only limiting factor, typical upper-bound ITRS-scaling speedups would still be limited to 15 \times . With conservative scaling, this best-case speedup is limited to 6.3 \times .

For the second experiment, we keep each application's parallelism at its real level and vary the power budget from 50W to 500W. As Figure 4(b) shows, eight of 12 benchmarks show no more than 10 \times speedup even with a practically unlimited power budget. That is, increasing core counts beyond a certain point does not improve performance due to the limited parallelism in the applications and the Amdahl's Law. Only four benchmarks have sufficient parallelism to even hypothetically sustain the exponential level of speedup.

Figure 4. Impact of application parallelism and power budget on speedup at 8nm.



The level of parallelism in PARSEC applications is the primary contributor to the dark silicon performance gap. However, in realistic settings the dark silicon resulting from power constraints limits the achievable speedup.

7.3. Core count projections

Different applications saturate performance improvements at different core counts. We consider the chip configuration that provides the best speedup for all of the applications as an *ideal* configuration. Figure 5 shows the number of cores (solid line) for the ideal CPU-like dynamic multicore configuration across technology generations. We choose the dynamic topology since it delivers the highest performance. The dashed line illustrates the number of cores required to achieve 90% of the ideal configuration's geometric mean speedup across PARSEC benchmarks. As depicted, with ITRS scaling, the ideal configuration integrates 442 cores at 8nm; however, 35 cores reach the 90% of the speedup achievable by 442 cores. With conservative scaling, the 90% speedup core count is 20 at 8nm.

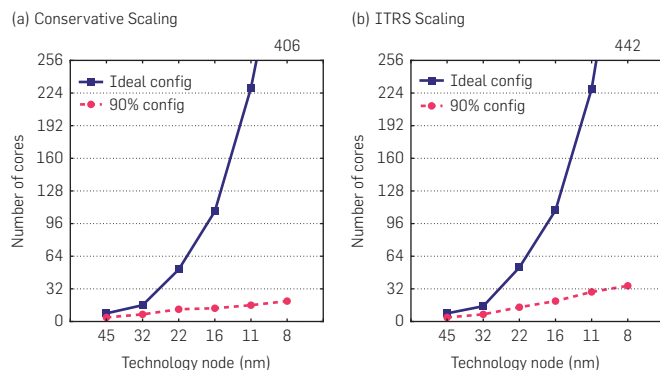
Due to limited parallelism in the PARSEC suite, even with novel heterogeneous topologies and optimistic ITRS scaling, integrating more than 35 cores improves performance only slightly for CPU-like topologies.

7.4. Sensitivity studies

Our analysis thus far examined typical configurations and showed poor scalability for the multicore designs. A natural question is, *can simple configuration changes (percentage cache area, memory bandwidth, etc.) provide significant benefits that can bridge the dark silicon gap?* We investigate three such simple changes: L2 cache size, memory bandwidth, and simultaneous multi-threading (SMT).

L2 cache area. Figure 6(a) shows the optimal speedup at 45nm as the amount of a symmetric CPU's chip area devoted to L2 cache varies from 0% to 100%. In this study

Figure 5. Number of cores for the ideal CPU-like dynamic multicore configurations and the number of cores delivering 90% of the speedup achievable by the ideal configurations across the PARSEC benchmarks.



we ignore any increase in L2 cache power or increase in L2 cache access latency. Across the PARSEC benchmarks, the optimal percentage of chip devoted to cache varies from 20% to 50% depending on benchmark memory access characteristics. Compared to the unified 30% cache area for all the applications, using each application's optimal cache area improves performance merely by at most 20% across all benchmarks.

Memory bandwidth. Figure 6(b) illustrates the sensitivity of PARSEC performance to the available memory bandwidth for symmetric GPU multicores at 45nm. As the memory bandwidth increases, the speedup improves since more threads can be fed with data; however, the benefits are limited by power and/or parallelism and in 10 out of 12 benchmarks speedups do not increase by more than 2x compared to the baseline, 200GB/s.

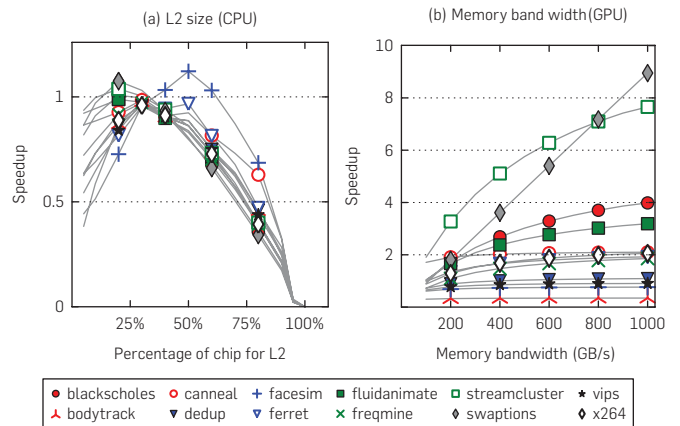
SMT. To simplify the discussion, we did not consider SMT support for the processors (cores) in the CPU multicore organization. SMT support can improve power efficiency of the cores for parallel workloads to some extent. We studied 2-way, 4-way, and 8-way SMT with no area or energy penalty, and observed that speedup improves with 2-way SMT by 1.5x in the best case and decreases as much as 0.6x in the worst case due to increased cache contention; the range for 8-way SMT is 0.3–2.5x.

8. ASSUMPTIONS AND LIMITATIONS

We discuss some of the important limitations of our model and argue that they do not significantly change our final results.

Dynamic voltage and frequency scaling (DVFS). Our device and core models do not explicitly consider dynamic voltage and frequency scaling; instead, we take an optimistic approach to account for their best-case settings. When deriving the Pareto frontiers, we set each processor to operate at its optimal voltage and frequency setting ($V_{dd,min}$, $Freq_{max}$). At a fixed V_{dd} setting, scaling down the frequency from $Freq_{max}$ results in a power/performance point inside the optimal Pareto curve, a suboptimal design point. However, scaling

Figure 6. Effect of L2 size and memory bandwidth on speedup at 45nm.



voltage up and operating at a new ($V'_{dd_{min}}, Freq'_{max}$) setting results in a different power-performance point that is still at the optimal frontier. Since we investigate all of the points along the frontier to find the optimal multicore configuration, our study covers multicore designs that induce heterogeneity to symmetric topologies through dynamic voltage and frequency scaling.

Architecture details in multicore model and validation.

The multicore model considers the first-order impact of caching, parallelism, and threading under assumptions that result only in optimistic projections (i.e., favorable multicore scaling). Comparing the output of the M-CMP model against published empirical results confirm that our model always overpredicts multicore performance. The model optimistically assumes that the workload is homogeneous, work is infinitely parallel during parallel sections of code, memory accesses never stall due to a previous access, and no thread synchronization, operating system serialization, or swapping occurs.

Server workloads. We do not directly study the server workloads, a domain where applications are highly concurrent and embarrassingly parallel. However, even in these types of workloads, resource scheduling and structural hazards such as competition for cache, memory bandwidth, DRAM storage, SSD IO, network IO, etc. limit parallelism. These factors induce a serial portion to the execution of the workloads. The key challenge is to measure the amount of serialization from these structural hazards, which is an interesting future study. Once the amount of serialization is measured, our models can be applied to the server workloads to project the amount of dark silicon and its effects. Generally, if the effective parallelism is less than 99%, the results suggest that dark silicon and its effects will manifest. Furthermore, Hardavellas et al. forecast the limits of multicore scaling and the emergence of dark silicon in servers with workloads that have an inherent abundance of parallelism.¹² They project that for server workloads such as online transaction processing (OLTP), decision support systems (DSS), and Web server (Apache), even with 3D-stacked memory, a significant amount of a multicore chip will be dark as technology scales. They determine that power and limited and non-scalable off-chip bandwidth are the primary limiting factors to multicore performance scaling and result in dark silicon for server workloads.

Alternative cores. We do not consider embedded ARM or Tiler cores in this work because they are designed for restricted domains and their SPECmark scores are not available for a meaningful comparison.

9. A PATH FORWARD?

For decades, Moore's Law *plus* Dennard scaling permitted more transistors, faster transistors, and more energy efficient transistors with each new process node, justifying the enormous costs required to develop each new process node. Dennard scaling's failure led industry to race down the multicore path, which for sometime permitted performance scaling for parallel and multitasked

workloads, allowing the economics of process scaling to hold. A key question for the computing community is whether scaling multicores will provide the performance and value needed to scale down many more technology generations. Are we in a long-term "multicore era," or will it instead be a "multicore decade" (2004–2014)? Will industry need to move in different, perhaps radical, directions to justify the cost of scaling? To answer the question, this paper models an upper bound on parallel application performance available from multicore and CMOS scaling—assuming no major disruptions in process scaling or core efficiency. Using a constant area and power budget, this study showed that the space of known multicore designs (CPU, GPU, their hybrids) or novel heterogeneous topologies (e.g., dynamic or composable) falls far short of the historical performance gains to which the microprocessor industry is accustomed. Even with aggressive ITRS scaling projections, scaling cores achieve a geometric mean $7.9\times$ speedup in 10 years at 8nm—a 23% annual gain. Our findings suggest that without process breakthroughs, directions beyond multicore are needed to provide performance scaling. There are reasons to be both pessimistic and optimistic.

9.1. Pessimistic view

A pessimistic interpretation of this study is that the performance improvements to which we have grown accustomed over the past 40 years are unlikely to continue with multicore scaling as the primary driver. The transition from multicore to a new approach is likely to be more disruptive than the transition to multicore. Furthermore, to sustain the current cadence of Moore's Law, the transition needs to be made in only a few years, much shorter than the traditional academic time frame for research and technology transfer. Major architecture breakthroughs in "alternative" directions such as neuromorphic computing, quantum computing, or bio-integration will require even more time to enter the industrial product cycle. Furthermore, while a slowing of Moore's Law will obviously not be fatal, it has significant economic implications for the semiconductor industry.

9.2. Optimistic view

Technology. The study shows if energy efficiency breakthroughs are made on supply voltage and process scaling, the performance improvement potential for multicore scaling is still high for applications with very high degrees of parallelism.

The need for microarchitecture innovations. Our study shows that fundamental processing limitations emanate from the processor core. The limited improvements on single-threaded performance is the inhibiting factor. Clearly, architectures that move well past the power/performance Pareto-optimal frontier of today's designs are necessary to bridge the dark silicon gap and utilize the increases in transistor count. Hence, improvements to the processor core efficiency will have significant impact on performance improvement and will enable technology scaling even though the core consumes


only 20% of the power budget for an entire laptop, smartphone, tablet, etc. When performance becomes limited, microarchitectural techniques that occasionally use parts of the chip to deliver outcomes orthogonal to performance, such as security, programmer productivity, and software maintainability are ways to sustain the economics of the industry. We believe this study will revitalize and trigger microarchitecture innovations, making the case for their urgency and their potential impact.

Efficiency through specialization. Recent work has quantified three orders of magnitude difference in efficiency between general-purpose processors and ASICs.¹¹ However, there is a well-known tension between efficiency and programmability. Designing ASICs for the massive base of quickly changing general-purpose applications is currently infeasible. Programmable accelerators, such as GPUs and FPGAs, and specialized hardware can provide an intermediate point between the efficiency of ASICs and the generality of conventional processors, gaining significant improvements for specific domains of applications. Even though there is an emerging consensus that specialization and acceleration is a promising approach for efficiently utilizing the growing number of transistors, developing programming abstractions that allow general-purpose applications to leverage specialized hardware and programmable accelerators remain challenging.

Opportunity for disruptive innovations. Our study is based on a model that takes into account properties of devices, processor cores, multicore organizations, and topologies. Thus the model inherently provides the areas to focus on for innovation. To surpass the dark silicon performance barrier highlighted by our work, designers must develop systems that use significantly more energy-efficient techniques. Some examples include device abstractions beyond digital logic (error-prone devices); processing paradigms beyond superscalar, SIMD, and SIMT; and program semantic abstractions allowing probabilistic and approximate computation. There is an emerging synergy between the applications that can tolerate approximation and the unreliability in the computation fabric as technology scales down. If done in a disciplined manner, relaxing the high tax of providing perfect accuracy at the device, circuit, and architecture level can provide a huge opportunity to improve performance and energy efficiency for the domains in which applications can tolerate approximate computation yet deliver acceptable outputs. Our results show that such radical departures are needed and the model provides quantitative measures to examine the impact of such techniques.

The model we have developed in the paper is useful to determine an optimal multicore configuration given a workload set, a power and area budget, and a technology generation. It can also be used to project expected multicore performance for the best configurations under a range of assumptions. We have made the models available for general use at the following URL: <http://research.cs.wisc.edu/vertical/DarkSilicon>.

Acknowledgments

We thank Shekhar Borkar for sharing his personal views on how CMOS devices are likely to scale. Support for this research was provided by NSF under the following grants: CCF-0845751, CCF-0917238, and CNS-0917213. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF. 

References

- Amdahl, G.M. Validity of the single processor approach to achieving large-scale computing capabilities. In *AFIPS* (1967).
- Azizi, O., Mahesri, A., Lee, B.C., Patel, S.J., Horowitz, M. Energy-performance tradeoffs in processor architecture and circuit design: A marginal cost analysis. In *ISCA* (2010).
- Bhadauria, M., Weaver, V., McKee, S. Understanding PARSEC performance on contemporary CMPs. In *IISWC* (2009).
- Bienia, C., Kumar, S., Singh, J.P., Li, K. The PARSEC benchmark suite: Characterization and architectural implications. In *PACT* (2008).
- Borkar, S. The exascale challenge. Keynote at *VLSI-DAT* (2010).
- Chakraborty, K. Over-provisioned multicore systems. PhD thesis, University of Wisconsin-Madison (2008).
- Chung, E.S., Milder, P.A., Hoe, J.C., Mai, K. Single-chip heterogeneous computing: Does the future include custom logic, FPGAs, and GPUs? In *MICRO* (2010).
- Dennard, R.H., Cai, J., Kumar, A. A perspective on today's scaling challenges and possible future directions. *Solid-State Electron.* 5, 4 (Apr. 2007).
- Dennard, R.H., Gaensslen, F.H., Rideout, V.L., Bassous, E., LeBlanc, A.R. Design of ion-implanted mosfet's with very small physical dimensions. *IEEE J. Solid-State Circuits* 9 (Oct. 1974).
- Guz, Z., Bolotin, E., Keidar, I., Kolodny, A., Mendelson, A., Weiser, U.C. Many-core vs. many-thread machines: Stay away from the valley. *IEEE Comput. Archit. Lett.* 8 (Jan. 2009).
- Hameed, R., Qadeer, W., Wachs, M., Azizi, O., Solomatnikov, A., Lee, B.C., Richardson, S., Kozyrakis, C., Horowitz, M. Understanding sources of inefficiency in general-purpose chips. In *ISCA* (2010).
- Hardavellas, N., Ferdman, M., Falsafi, B., Ailamaki, A. Toward dark silicon in servers. *IEEE Micro* 31, 4 (Jul.-Aug. 2011).
- Hempstead, M., Wei, G.Y., Brooks, D. Navigo: An early-stage model to study power-constrained architectures and specialization. In *MoBS* (2009).
- Hill, M.D., Marty, M.R. Amdahl's law in the multicore era. *Computer* 41, 7 (Jul. 2008).
- Ipek, E., Kirman, M., Kirman, N., Martinez, J.F. Core fusion: Accommodating software diversity in chip multiprocessors. In *ISCA* (2007).
- ITRS. International technology roadmap for semiconductors, the 2010 update. <http://www.itrs.net> (2011).
- Kim, C., Sethumadhavan, S., Govindan, M.S., Ranganathan, N., Gulati, D., Burger, D., Keckler, S.W. Composable lightweight processors. In *MICRO* (2007).
- Moore, G.E. Cramming more components onto integrated circuits. *Electronics* 38, 8 (Apr. 1965).
- Pollack, F. New microarchitecture challenges in the coming generations of CMOS process technologies. Keynote at *MICRO* (1999).
- SPEC. Standard performance evaluation corporation. <http://www.spec.org> (2011).
- Suleman, A.M., Mutlu, O., Qureshi, M.K., Patt, Y.N. Accelerating critical section execution with asymmetric multi-core architectures. In *ASPLOS* (2009).
- Venkatesh, G., Sampson, J., Goulding, N., Garcia, S., Bryksin, V., Lugo-Martinez, J., Swanson, S., Taylor, M.B. Conservation cores: Reducing the energy of mature computations. In *ASPLOS* (2010).

Hadi Esmailzadeh (hadiane@cs.washington.edu), University of Washington.

Emily Blem (blem@cs.wisc.edu), University of Wisconsin-Madison.

Renée St. Amant (stamant@cs.utexas.edu), The University of Texas at Austin.

Karthikeyan Sankaralingam (karu@cs.wisc.edu), University of Wisconsin-Madison.

Doug Burger (dburger@microsoft.com), Microsoft Research.

Bucknell University Assistant Professor, Computer Science – Computer Architecture

Applications are invited for a tenure-track position in computer science beginning mid-August 2013. We expect to hire at the Assistant Professor level, but outstanding candidates will be considered at Associate Professor or Professor; years of credit toward tenure will be awarded based upon qualifications. We seek a teacher-scholar with a demonstrated ability to work with a diverse student body and are specifically interested in candidates whose research area is in computer architecture. This position is responsible for the department's upper-level undergraduate Computer Architecture course. In addition, the successful candidate must be able to participate in the teaching of required core courses and be able to develop elective courses in the candidate's area of expertise. Candidates are expected to have completed or be in the final stages of completing their Ph.D. by the beginning of the 2013 fall semester. A strong commitment to excellence in teaching and scholarship is required.

Bucknell is a highly selective private university emphasizing quality undergraduate education in engineering and in liberal arts and sciences. The B.S. programs in computer science are ABET accredited. The computing environment is Linux/Unix-based. More information about the department can be found at:

<http://www.bucknell.edu/ComputerScience/>

Applications will be considered as received and recruiting will continue until the position is filled. Candidates are asked to submit a cover letter, CV, a statement of teaching philosophy and research interests, and the contact information for three references. Please submit your application to

<http://jobs.bucknell.edu/>

by searching for the "Computer Science Faculty Position – Computer Architecture".

Please direct any questions to Professor Stephen Guattery of the Computer Science Department at guattery@bucknell.edu.

Bucknell University, an Equal Opportunity Employer, believes that students learn best in a diverse, inclusive community and is therefore committed to academic excellence through diversity in its faculty, staff, and students. Thus, we seek candidates who are committed to Bucknell's efforts to create a climate that fosters the growth and development of a diverse student body. We welcome applications from members of groups that have been historically underrepresented in higher education.

Carnegie Mellon University Computer Science Department Teaching Track Positions

Applications are invited for two teaching-track positions in Computer Science, beginning Fall 2013. This is a renewable, career-oriented position with an initial appointment for three years. We seek highly qualified applicants with a strong commitment to excellence in teaching and the ability to teach at all levels in the undergraduate curriculum.

Applicants for the position must have a Ph.D. in Computer Science or a related field, and demonstrated excellence in teaching Computer Science courses. Teaching-track appointments are typically at the rank of Assistant Teaching Professor, with the possibility of promotion to the ranks of Associate Teaching Professor and Teaching Professor. None of these ranks are tenured; applicants seeking a tenure-track position at a research university are therefore not a good match for these positions.

In order to receive full consideration, applicants should submit a letter of application, curriculum vitae, a statement of teaching philosophy, and the names and email addresses of three or more individuals whom the applicant has asked to provide letters of reference. Applicants should arrange for reference letters to be sent directly to the contact below. This information should be sent by February 28, 2013, to the contact listed below.

Additionally, applicants are encouraged to submit a video sample of their teaching. This enables applicants to add another dimension to their application. Since the people who will eventually fill these positions will be expected to be excellent classroom teachers, the video sample is an opportunity for candidates to show off their talents in a way other than traditional on paper means.

Please send your applications and accompanying materials to

Dr. Klaus Sutner
Computer Science Department
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213
email: sutner@cs.cmu.edu

Carnegie Mellon is an affirmative action/equal opportunity employer and we invite and encourage applications from women and underrepresented minorities.

Connecticut College Data Mining Postdoctoral / Visiting Faculty

Connecticut College is seeking candidates with research interests in the use of data mining/machine learning for analyzing biological data. See details at cs.conncoll.edu/bioinformatics.htm for

this faculty position to begin in August. Contact: Gary Parker. Email: bioinfosearch@conncoll.edu. Apply URL: <http://cs.conncoll.edu/bioinformatics.htm>

Dalhousie University Halifax, Canada Faculty of Computer Science Probationary Tenure Track Assistant Professor

Probationary Tenure Track Assistant Professor position in the Faculty of Computer Science

Dalhousie University (<http://www.dal.ca>) invites applications for a Probationary Tenure Track position at the Assistant Professor level in the Faculty of Computer Science (<http://www.cs.dal.ca>) that currently has 30 faculty members, approximately 425 undergraduate majors and 240 master's and doctoral students. The Faculty partners with other Faculties in the University to offer the Master of Electronic Commerce, Master of Health Informatics and Master of Science in Bioinformatics programs, and is an active participant in the Interdisciplinary PhD program.

Dalhousie University is located in Halifax, Nova Scotia (<http://www.halifaxinfo.com/>), which is the largest city in Atlantic Canada and affords its residents outstanding quality of life.

The Faculty welcomes applications from outstanding candidates in Computer Science. An applicant should have a PhD in Computer Science or related area and be comfortable teaching core computer science courses, particularly Software Engineering. Evidence of a strong commitment to and aptitude for research and teaching is essential. The ideal candidate will be open to collaborative research within the faculty and add to or complement existing research strengths and strategic research directions of the Faculty.

Applications should include an application letter, curriculum vitae, a statement of research and teaching interests, sample publications, and the names, email addresses and physical addresses of three referees. The application must include the Equity Self-Identification form (see the URL below). All documents are to be submitted to the email address below as PDF files.

Applicants should provide their referees with the URL of this advertisement (see below), and request that they forward letters of reference by email to the same address.

Applications will be accepted until April 30, 2013

All qualified candidates are encouraged to apply; however Canadian and permanent residents will be given priority. Dalhousie University is an Employment Equity/Affirmative Action Employer. The University encourages applications from qualified Aboriginal people, persons with a disability, racially visible persons and women.

Submission Address for application documents and reference letters: appointments@cs.dal.ca

Location of this advertisement:
www.cs.dal.ca

Self-Identification form (PDF):

http://hrehp.dal.ca/Files/Academic_Hiring_%28For/selfid02.pdf

Self-Identification form (Word):

http://hrehp.dal.ca/Files/Academic_Hiring_%28For/selfid02.doc

Grove City College Professor of Computer Science

Grove City College announces a faculty opening in the Computer Science Department in July 2013. A Ph.D. in Computer Science is required. Responsibilities include teaching lower-level and upper-level Computer Science courses, guiding undergraduate-research projects, and developing innovative pedagogy.

Rank and salary are commensurate with qualifications. Grove City College is a highly selective college of liberal arts, sciences, and engineering where intellectual inquiry remains open to the questions religion raises and affirms the answers Christianity offers.

Send letter of application, vita, transcripts, names of four references (three professional and one pastoral), a brief statement of how you would engage undergraduates in your research plans, and a brief essay relating your philosophy of a liberal arts education and teaching Computer Science to the College's mission (see www.gcc.edu) to: William P. Anderson, Jr., Ph.D., Provost and VP for

Academic Affairs, 100 Campus Drive, Grove City, PA 16127 or electronically to laklaiber@gcc.edu.

Grove City College is a private educational institution noted for its academic excellence where scholarship is informed by Christian principles. It does not discriminate on the basis of age, race, color, creed, sex, marital status, disability, or national or ethnic origin in the administration of its educational policies, admission policies, scholarship and loan programs, and athletic and other college-administrative programs.

Harvard University School of Engineering and Applied Sciences Preceptor in Computer Science

The Harvard School of Engineering and Applied Sciences (SEAS) seeks applicants for the position of Preceptor in Computer Science. The preceptor will be primarily responsible for the coordination and support of core undergraduate courses in Computer Science, including Computer Science 50: Introduction to Computer Science I, CS51: Introduction to Computer Science II, and CS61: Systems Programming and Machine Organization. The position is an annual (twelve-month), academic appointment, renewable for up to three or more years, depending on continuing curricular need and performance.

Typical Responsibilities:

- ▶ Work with and report to the faculty members who are the principal course instructors on the preparation of lecture materials, in-class exercises, homework assignments, and examination questions

- ▶ Lead weekly sections and teach undergraduate students

- ▶ Hire, train, and supervise a team of teaching fellows, including leading regular TF meetings

- ▶ Design and implement tools and test cases to be used by each courses' students and TFs

- ▶ Hold office hours and advise undergraduate students across the Computer Science concentration

A background in computer science is required; master's or PhD in computer science or education preferred. We seek candidates with superior organizational, written and interpersonal communication skills, along with the ability to lead and train teaching fellows and to manage relationships with hundreds of undergraduate students. Prior experience teaching and designing problem sets and classroom exercises is strongly preferred and industry experience is a plus.

Applicants must apply on-line at <http://academicpositions.harvard.edu/postings/4473>. Required documents include a cover letter, CV, names and contact information for at least three references, and a summary of prior teaching or tutoring experience including, for each course, its name, school, description, syllabus (if available), and website (if available).

Applications will be reviewed as they are received. Applicants are strongly encouraged to submit applications by **February 15, 2013**. However, applications will continue to be accepted until the position is filled. Harvard is an Equal Opportunity/Affirmative Action Employer. Applications from women and minority candidates are strongly encouraged.



Yale School of Engineering & Applied Science Department of Electrical Engineering

Junior Search in Communications and Networking at Yale University

Yale University's Electrical Engineering Department invites applications from qualified individuals for a tenure-track, non-tenured faculty position in the area of communications and networking. Subfields of interest include wireless communications, networking, signal processing, network optimization, network economics, machine learning, and network science. All candidates should be strongly committed to both teaching and research and should be open to collaborative research. Candidates should have distinguished records of research accomplishments and should be willing and able to participate in shaping Yale's expanding program in electrical engineering. Yale University is an Affirmative Action/Equal Opportunity Employer. Yale values diversity among its students, staff, and faculty and strongly welcomes applications from women and under represented minorities. The review process will begin November 15, 2012. Applicants should include a CV, a research statement, a teaching statement and submit to <http://academicjobsonline.org/>.

Senior Search in Communications and Networking at Yale University

Yale University's Electrical Engineering Department invites applications from qualified individuals for a tenured faculty position in the area of communications and networking. Subfields of interest include wireless communications, networking, signal processing, network optimization, network economics, machine learning, and network science. All candidates should be strongly committed to both teaching and research and should be open to collaborative research. Candidates should have distinguished records of research accomplishments and should be willing and able to take the lead in shaping Yale's expanding program in electrical engineering. Yale University is an Affirmative Action/Equal Opportunity Employer. Yale values diversity among its students, staff, and faculty and strongly welcomes applications from women and under represented minorities. The review process will begin November 15, 2012. Applicants should include a CV, a research statement, a teaching statement and submit to <http://academicjobsonline.org/>.

Senior Position in Computer Engineering at Yale University

Yale University's Electrical Engineering Department invites applications from qualified individuals for a tenured faculty position in computer engineering. Subfields of interest include systems on a chip, embedded systems, VLSI, design automation, energy-efficient computing, low-power circuits, verification, networked systems, mobile computing, sensor networks, and biodevices. All candidates should be strongly committed to both teaching and research and should be open to collaborative research. Candidates should have distinguished records of research accomplishments and should be willing and able to take the lead in shaping Yale's expanding program in computer engineering. Yale University is an Affirmative Action/Equal Opportunity Employer. Yale values diversity among its students, staff, and faculty and strongly welcomes applications from women and under represented minorities. The review process will begin on November 15, 2012. Applicants should include a CV, a research statement, a teaching statement and submit to <http://academicjobsonline.org/>.



WE ARE HIRING!

We have openings for Research Scientists, Applied Scientists and Research Engineers at our locations in the US, Spain, Israel, India and China. Come join us in solving real world problems at scale, diving into oceans of data, creating new products and experiences, and collaborating in ground-breaking research. We are looking for scientists in many disciplines of Computer Science such as Machine Learning, Natural Language Processing, Statistical Data Analysis, Systems, Computational Advertising, Optimization, Media, Human-Computer Interaction and Mobile Experiences.

For our Research and Applied Scientist positions, we are looking for full time and postdoc scientists, as well as interns for summer 2013.

For more information and to apply please visit

<http://careers.yahoo.com>

and search for scientist positions.



Interns, please apply here:

<http://y.ahoo.it/uhMpv>



Come make your mark on science – come grow with us!

About Yahoo! Labs

Founded in 2005, Yahoo! Labs pioneered important research in web mining and systems, and created the field of Computational Advertising. Today we continue to be a leader in industrial research. As the center of scientific excellence at Yahoo! we deliver both fundamental and applied scientific leadership, publish research, and create new technologies that power Yahoo!'s products and experiences. We're responsible for big inventions, and our goals are nothing short of inventing the future of the Internet and creating the next generation of businesses for Yahoo!.

Kettering University Assistant Professor of Computer Science

The Computer Science Department at Kettering University seeks outstanding applicants for a tenure-track position at the rank of assistant professor, beginning in July 2013. A PhD in Computer Science is required. We have been named as a US Ignite institution and are currently building the high-speed broadband infrastructure necessary to participate in Internet-scale networking experimentation across university campuses and cities. The successful candidate will be able to foster academic/public/private collaborations in the areas of software-defined networks, cloud computing, and wireless networking, and will have experience, or can demonstrate competency, in teaching a broad range of undergraduate computer science courses. Additional research interests in systems, security, or software engineering is considered an asset. Interested individuals must apply for the position on-line <https://jobs.kettering.edu>. Applications will be reviewed beginning January 7, 2013 and continue until the position is filled. Kettering University is committed to excellence through diversity in its faculty, staff, and students. AA/EOE

Max Planck Institute for Informatics Junior Research Group Leader

The Max Planck Institute for Informatics, as the coordinator of the Max Planck Center for Visual Computing and Communication (MPC-VCC),

invites applications for **Junior Research Groups Leaders** in the Max Planck Center for Visual Computing and Communication

The Max Planck Center for Visual Computing and Communications offers young scientists in information technology the opportunity to develop their own research program addressing important problems in areas such as image communication, computer graphics, geometric computing, imaging systems, computer vision, human machine interface, distributed multimedia architectures, multimedia networking, visual media security.

The center includes an outstanding group of faculty members at Stanford's Computer Science and Electrical Engineering Departments, the Max Planck Institute for Informatics, and Saarland University.

The program begins with a preparatory 1-2 year postdoc phase (Phase P) at the Max Planck Institute for Informatics, followed by a two-year appointment at Stanford University (Phase I) as a visiting assistant professor, and then a position at the Max Planck Institute for Informatics as a junior research group leader (Phase II). However, the program can be entered flexibly at each phase, commensurate with the experience of the applicant.

Applicants to the program must have completed an outstanding PhD. Exact duration of the preparatory postdoc phase is flexible, but we typically expect this to be about 1-2 years. Applicants who completed their PhD in Germany may enter Phase I of the program directly. Applicants for Phase II are expected to have completed a post-

doc stay abroad and must have demonstrated their outstanding research potential and ability to successfully lead a research group.

Reviewing of applications will commence on 01 Jan 2013. The final deadline is 31 Jan 2013. Applicants should submit their CV, copies of their school and university reports, list of publications, reprints of five selected publications, names of references, a brief description of their previous research and a detailed description of the proposed research project (including possible opportunities for collaboration with existing research groups at Saarbrücken and Stanford) to:

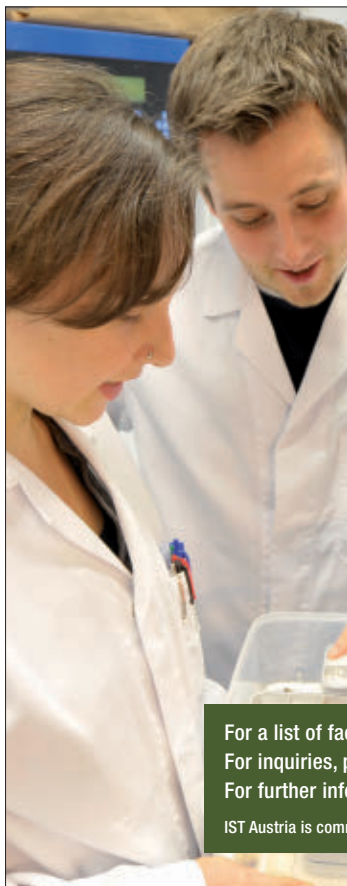
Prof. Dr. Hans-Peter Seidel
Max Planck Institute for Informatics,
Campus E 1 4, 66123 Saarbrücken, Germany;
Email: mpc-vc@mpi-inf.mpg.de

The Max Planck Center is an equal opportunity employer and women are encouraged to apply.

Additional information is available on the website <http://www.mpc-vc.de>

Missouri University of Science and Technology (S&T) Computer Science Department Chair

The Department of Computer Science at the Missouri University of Science and Technology (S&T) invites applications for the position of Department Chair starting Fall 2013. The successful candidate will assume a leadership role in the department to strengthen and expand the depart-



ISTFELLOW

CALL FOR POSTDOCTORAL FELLOWS

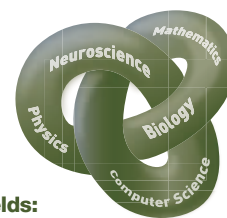
IST Austria has set up a program for exceptional postdoctoral fellows with an emphasis on interdisciplinary work. Appointments will be for 2–4 years. Applications will be accepted at any time, but fellows will be selected twice a year in April and October, with deadlines on 15th of March and September, respectively. Applicants must have the support of one or more members of the IST Austria faculty.

Benefits:

- Internationally competitive salary
- Full social security coverage
- Travel, mobility and family allowance
- Funding for conferences and scientific visits

The institute offers postdoctoral positions in the following fields:
Biology | Computer Science | Mathematics | Physics | Neuroscience

ISTFELLOW is partially funded by the European Union.



For a list of faculty members please visit www.ist.ac.at.

For inquiries, please contact istfellow@ist.ac.at.

For further information, please refer to the ISTFELLOW website: <http://ist.ac.at/istfellow>

IST Austria is committed to Equality and Diversity.





جامعة الملك عبد الله
للعلوم والتقنية
King Abdullah University of
Science and Technology

King Abdullah University of Science and Technology (KAUST) invites applications for faculty positions at the rank of Full, Associate or Assistant Professor in the area of Computer Science.

KAUST, located on the Red Sea coast of Saudi Arabia, is an international graduate-level research university dedicated to advancing science and technology through bold and collaborative research and to addressing challenges of regional and global significance, thereby serving the Kingdom, the region, and the world. Newly opened in September 2009, KAUST is an independent and merit-based university and welcomes exceptional faculty, researchers, and students from around the world. KAUST is committed to cutting-edge research in the globally significant areas of water, food, energy and the environment. In addition, KAUST emphasizes research on the discipline of Computational Science and Engineering serving as an enabling technology for all its research activities. Areas of interest are:

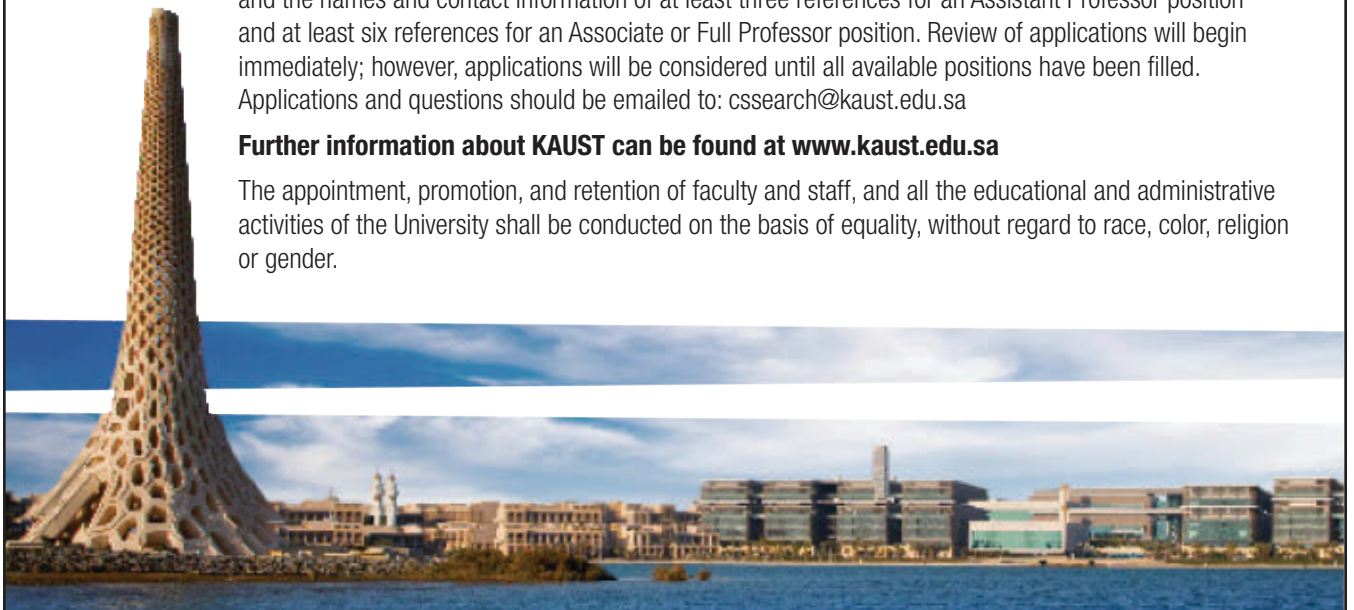
- **Management of very large data**
- **Data mining and knowledge extraction**
- **Parallel and distributed systems**
- **Data security**

High priority will be given to the overall originality and promise of the candidate's work rather than the candidate's sub-area of specialization within Computer Science. An earned PhD in Computer Science, Computer Engineering, Electrical Engineering, or a related field, and strong publication record are required. A successful candidate will be expected to teach courses at the graduate level and to build and lead a research group of postdoctoral fellows and graduate students. Faculty members enjoy secure research funding from KAUST and have opportunities for additional funding through several KAUST provided sources and through industry collaborations.

Applications submitted as a single PDF/Word file should include a cover letter indicating the position of interest, a curriculum vitae with a list of publications, statements of research, and teaching interests, and the names and contact information of at least three references for an Assistant Professor position and at least six references for an Associate or Full Professor position. Review of applications will begin immediately; however, applications will be considered until all available positions have been filled. Applications and questions should be emailed to: cssearch@kaust.edu.sa

Further information about KAUST can be found at www.kaust.edu.sa

The appointment, promotion, and retention of faculty and staff, and all the educational and administrative activities of the University shall be conducted on the basis of equality, without regard to race, color, religion or gender.



ment's research and teaching missions. Candidates must, by the date of appointment, have the qualifications and standing to be appointed as a tenured full professor. The candidate should have a strong record of research and teaching at a university, but candidates from industry with a strong research record coupled with academic and administrative experience are encouraged to apply.

The department's current research strengths are in software engineering, mobile and distributed systems, cyber security and information assurance, social-cyber-physical systems and critical infrastructure protection, computational intelligence, pervasive computing, and wireless and sensor networks. Recent funding sources include NSF, NSA, ARL, AFRL, NIST, Sandia National Laboratories, as well as the U.S. Departments of Energy, Defense, and Education. The department's visions and strategic plan, activities, and research well as required and desired Attributes, Skills, and Characteristics are detailed further on our web site: <http://cs.mst.edu>.

The department has 18 full-time faculty lines as well as some joint appointments and adjunct faculty, and grants BS, MS and Ph.D. degrees as well as graduate certificates and a minor. The program is ABET accredited. As of fall semester 2012, the department has over 350 BS, 50 MS, and 30 Ph.D. students. The department leads two centers and many of our faculty members participate in interdisciplinary research with several campus research centers and departments.

Details of the application process are given at <http://hraadi.mst.edu/hr/employment/>. Review of applications will begin February 15, 2013. Ap-

plications received after April 1, 2013 will not be given consideration. For more information prior to submitting an application, please contact the Search Committee Chair, Dr. Bruce McMillin, at ff@mst.edu or the Search Committee Vice-Chair, Dr. Sanjay Madria, at madrias@mst.edu.

All submitted application materials must have the position reference number in order to be processed. Acceptable electronic formats that can be used for email attachments include PDF and Word; hardcopy application materials will not be accepted.

Missouri S&T is an AA/EQ Employer and does not discriminate based on race, color, religion, sex, sexual orientation, national origin, age, disability, or status as Vietnam-era veteran. Females, minorities, and persons with disabilities are encouraged to apply. Missouri S&T is responsive to the needs of dual-career couples.

North Carolina State University Department of Computer Science Faculty Position Assistant/Associate/Full Professor

The Department of Computer Science at North Carolina State University (NCSU) seeks to fill a tenure-track faculty position in the area of Software Engineering starting August 16, 2013. Software engineering candidates with research experience in requirements engineering are especially encouraged to apply.

Successful candidate must have a strong commitment to academic and research excellence,

and an outstanding research record commensurate with the expectations of a major research university. Required credentials include a doctorate in Computer Science or a related field. While the department expects to hire at the Assistant Professor level, candidates with exceptional research records are encouraged to apply for a senior position. The department is one of the largest and oldest in the country. It is part of NCSU's College of Engineering. The department's research expenditures and recognition have been growing steadily. For example, we have one of the largest concentrations in the country of prestigious NSF Early Career Award winners (total of 21).

NCSU is located in Raleigh, the capital of North Carolina, which forms one vertex of the world-famous Research Triangle Park (RTP). RTP is an innovative environment, both as a metropolitan area with one of the most diverse industrial bases in the world, and as a center of excellence promoting technology and science. The Research Triangle area is routinely recognized in nationwide surveys as one of the best places to live in the U.S. We enjoy outstanding public schools, affordable housing, and great weather, all in the proximity to the mountains and the seashore.

Applications will be reviewed as they are received. The positions will remain open until suitable candidates are identified. Applicants should submit the following materials online at <http://jobs.ncsu.edu> (reference position number 1092) cover letter, curriculum vitae, research statement, teaching statement, and names and complete contact information of four references, including email addresses and phone numbers.



TEMASEK RESEARCH FELLOWSHIP (TRF)

A globally connected cosmopolitan city, Singapore provides a supportive environment for a vibrant research culture. Its universities Nanyang Technological University (NTU), National University of Singapore (NUS) and Singapore University of Technology and Design (SUTD) invite outstanding young researchers to apply for the prestigious TRF awards.

Under the TRF scheme, selected young researchers with a PhD degree have an opportunity to conduct and lead defence-related research. It offers:

- A 3-year research grant of up to S\$1 million commensurate with the scope of work, with an option to extend for another 3 years
- Postdoctoral or tenure-track appointment (eligibility for tenure-track will be determined by the university)
- Attractive and competitive remuneration

Fellows may lead, conduct research and publish in these areas:

- Advanced Materials for Aerospace Applications
- Bio-mimetic Aerodynamics
- Cognitive Science and Neuroengineering
- Cyber Security
- High Power Laser Diode
- High Speed High Voltage Switching Devices

For more information and application procedure, please visit:

NTU – http://www3.ntu.edu.sg/trf/index_trf.html
NUS – <http://www.nus.edu.sg/dpr/funding/trf.html>
SUTD – <http://www.sutd.edu.sg/trf>

Closing date: 15 March 2013 (Friday)

Shortlisted candidates will be invited to Singapore to present their research plans, meet local researchers and identify potential collaborators in July 2013.

Northwestern University

Assistant Professor in Database Systems

The Department of Electrical Engineering and Computer Science at Northwestern University invites applications for a tenure-track assistant professor position in database systems to start in fall 2013. We are interested in exceptional candidates in all areas of database systems, but have a particular focus on areas such as large-scale data management, integration of structured and unstructured data, parallel and distributed data mining and analytics, stream databases, and database engines for scalable computing and emerging computer architectures.

A Ph.D. in Computer Science or Computer Engineering is required, as is a clear track record of success in database systems. Successful candidates will be expected to carry out world class-research, collaborate with other faculty, and teach effectively at the undergraduate and graduate levels. Compensation and start-up packages are negotiable and will be competitive.

Northwestern EECS consists of over 50 faculty members of international prominence whose interests span a wide range. Northwestern University is located in Evanston, Illinois on the shores of Lake Michigan just north of Chicago. Further information about the Department and the University is available at <http://www.eecs.northwestern.edu> and <http://www.northwestern.edu>.

To ensure full consideration, applications should be received by **February 15, 2013**, but applications will be accepted until the position is filled.

To apply, first read full upload instructions at <http://eecs.northwestern.edu/academic-openings.html>. Applicants will submit (1) a cover letter, (2) a curriculum vitae, (3) statements of research and teaching interests, (4) three representative publications, and (5) at least three, but no more than five references. For general questions about the search or application assistance post submission, contact db-search@eecs.northwestern.edu.

The aforementioned application materials may also be sent to: **Database Systems Faculty Search Committee, Department of Electrical Engineering and Computer Science, Technological Institute, L359, Northwestern University, 2145 Sheridan Road, Evanston, IL 60208, USA.**

Northwestern University is an equal opportunity, affirmative action employer. Qualified women and minorities are encouraged to apply. It is the policy of Northwestern University not to discriminate against any individual on the basis of race, color, religion, national origin, gender, sexual orientation, marital status, age, disability, citizenship, veteran status, or other protected group status. Hiring is contingent upon eligibility to work in the United States.

Candidates can obtain information about the department and its research programs, as well as more detail about the position advertised here at <http://www.csc.ncsu.edu/>. Inquiries may be sent via email to: facultyhire@csc.ncsu.edu.

NCSU is an equal opportunity and affirmative action employer. In addition, NCSU welcomes all persons without regard to sexual orientation or genetic information. Individuals with disabilities desiring accommodations in the application process should contact the Department of Computer Science at (919) 515-2858.

Palo Alto Research Center (PARC, a Xerox Company)
Research Scientist / Senior Research Scientist in Security

PARC - Research Scientist/Senior Research Scientist in Security Candidates in all areas of cyber security will be considered, with particular interest in: systems and network security, security in cloud computing and ubiquitous environments, machine learning & security, applied cryptography.

Apply at:
<http://www.parc.com/about/careers/>

Princeton University
Computer Science
Part-Time or Full-Time Lecturer

The Department of Computer Science seeks applications from outstanding teachers to assist the faculty in teaching our introductory course sequence or some of our upper-level courses starting February 1, 2013.

Depending on the qualifications and interests of the applicant, the job responsibilities will include such activities as teaching recitation sections and supervising graduate-student teaching assistants; grading problem sets and programming assignments, and supervising students in the grading of problem sets and programming assignments; developing and maintaining online curricular material, classroom demonstrations, and laboratory exercises; and supervising undergraduate research projects. An advanced degree in computer science, or related field, is required (PhD preferred).

The position is for one semester with possibility of renewal for 1-year terms, up to six years, depending upon departmental need.

Princeton University is an equal opportunity employer and complies with applicable EEO and affirmative action regulations. You may apply online, by submitting a letter of application, resume and names of three references at <http://jobs.cs.princeton.edu/lecturer>. (Princeton University Requisition number 1200848)

Swarthmore College
Tenure Track Assistant Professor

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.

Applications are invited for a tenure track position at the assistant professor level beginning Fall semester 2013. Swarthmore College is a small, selective, liberal arts college located 10 miles outside of Philadelphia. The Computer Science Department offers majors and minors at the undergraduate level. Applicants must have teaching experience and should be comfortable teaching a wide range of courses at the introductory and intermediate level. Candidates should additionally have a strong commitment to involving undergraduates in their research. A Ph.D. in CS by or near the time of appointment is required. We are particularly interested in applicants that add breadth to our department, including the areas of databases, networking, security, theory, compilers, and programming languages. Strong applicants in other areas will also be considered.

Priority will be given to applications received by December 15, but will be accepted until the position is filled. Applications should include a vita, teaching statement, research statement, and three letters of reference, at least two that speak to the candidate's teaching ability.

Apply for this Job:

Contact Person: Richard Wicentowski
Email Address:
jobs2013@cs.swarthmore.edu
Phone: 610-328-8272
Fax: 610-328-8606
Apply URL:
<http://goo.gl/LPYF2>

University of Northern Iowa
Assistant Professor of Computer Science

The Department of Computer Science at the University of Northern Iowa invites applications for a tenure-track assistant professor position to begin August 2013. Applicants must hold a Ph.D. in Computer Science or a closely-related discipline. The department seeks candidates able to participate widely in the CS curriculum and conduct a research program involving undergraduates.

Detailed information about the position and the department are available at <http://www.cs.uni.edu/>

To apply, visit <http://jobs.uni.edu/>. Applications received by January 15, 2013, will be given full consideration. EOE/AA. Pre-employment background checks are required. UNI is a smoke-free campus.

University of Pittsburgh
School of Information Sciences
Professor of Practice

The School of Information Sciences (<http://www.ischool.pitt.edu>) at the University of Pittsburgh is seeking candidates for a **Professor of Practice at an assistant/associate/full professor level - Position #02025** (Non-tenure stream) to start in the fall term of 2013. The primary areas of interest include:

- ▶ Object-oriented systems analysis and design
- ▶ Information systems architecture
- ▶ Value-centered design



ADVERTISING IN CAREER OPPORTUNITIES

How to Submit a Classified Line Ad: Send an e-mail to acmm mediasales@acm.org. Please include text, and indicate the issue/or issues where the ad will appear, and a contact name and number.

Estimates: An insertion order will then be e-mailed back to you. The ad will be typeset according to CACM guidelines. NO PROOFS can be sent. Classified line ads are NOT commissionable.

Rates: \$325.00 for six lines of text, 40 characters per line. \$32.50 for each additional line after the first six. The MINIMUM is six lines.

Deadlines: 20th of the month/2 months prior to issue date. For latest deadline info, please contact:

acmm mediasales@acm.org

Career Opportunities Online: Classified and recruitment display ads receive a free duplicate listing on our website at:

<http://jobs.acm.org>

Ads are listed for a period of 30 days.

For More Information Contact:

**ACM Media Sales
at 212-626-0686 or
acmm mediasales@acm.org**

ACM Transactions on Reconfigurable Technology and Systems



This quarterly publication is a peer-reviewed and archival journal that covers reconfigurable technology, systems, and applications on reconfigurable computers. Topics include all levels of reconfigurable system abstractions and all aspects of reconfigurable technology including platforms, programming environments and application successes.

www.acm.org/trets
www.acm.org/subscribe



The undergraduate program prepares students become system designers, system analysts, database managers, system administrators, programmer analysts, network engineers, and a host of related jobs. A clinical faculty member with experience in the design and implementation of small, large and distributed systems would serve as an excellent resource in the teaching of state-of-the-art industry practices.

For a complete description, please visit
<http://www.ischool.pitt.edu/news/facultyopenings.php>
 Contact: Search Committee
 Email: sissearch@sis.pitt.edu
 Phone: 412-624-5129
 Fax: 412-624-5231
 Apply URL: <http://www.ischool.pitt.edu/news/facultyopenings.php>

The University of Pittsburgh is an Equal Opportunity, Affirmative Action employer and strongly encourages women and candidates from under-represented minorities to apply.

University of São Paulo Institute of Mathematics and Statistics Department of Computer Science Tenure-Track Positions Assistant Professor

The Institute of Mathematics and Statistics of the University of São Paulo (IME-USP) invites applications for faculty positions at the Assistant Professor level. The Department is accepting applications in all areas of Computer Science.

We expect candidates with strong potential for research and teaching ability. The candidates should have a PhD in Computer Science or a related area. The selected candidate will be responsible for

developing research and for teaching to the programs of the department (undergraduate and graduate courses). Deadlines and documents required for the application are specified at www.ime.usp.br/dcc/faculty_position. The documents and selection interview may be either in Portuguese or English.

The University of São Paulo - USP is one of the most prestigious educational institution in South America. It is the best ranked Brazilian university. The Department of Computer Science of the IME-USP is responsible for the BSc, MSc and PhD degrees in Computer Science, offering some of the most competitive courses in Brazil.

More information:
<http://www.ime.usp.br/dcc>
 Contact: mac@ime.usp.br

Virginia Commonwealth University – VCU Computer Science VCU School of Engineering Assistant/Associate/Professor tenure-track

The Computer Science Department at Virginia Commonwealth University (VCU) invites applications for a tenure-track/tenured position at the rank of Assistant/Associate Professor. Outstanding well-funded candidates at higher level would also be considered. Candidates must have a Ph.D. in computer science, or in the related area. Junior faculty will be required to have an established research agenda and a clear potential for external funding, and potential for scholarship or creative expression to complement and expand existing expertise in the department and the School of Engineering, especially in the field of cyber security, broadly defined. For Associate/Professor level, faculty member will be required to have a well developed scholarly/research portfolio with evidence of multidisciplinary applications and external funding appropriate to complement and expand existing expertise within the department, especially in the field of cyber security.

Successful candidates are expected to teach courses in Computer Science at both the undergraduate and graduate level. Additionally, candidate must have demonstrated experience working in and fostering a diverse faculty, staff, and student environment or commitment to do so as a faculty member at VCU is required.

VCU, the largest urban university in Virginia, is a Carnegie research I extensive institution ranked in the top 100 universities in the U.S. in federal R&D expenditures, with a richly diverse community and commitment to multicultural opportunities.

For best consideration, applications should be submitted by March 1, 2013.

Candidates are to submit applications electronically to cmssearch@vcu.edu as a single pdf file that includes (in this order) a cover letter, resume, research and teaching statement, and the names and e-mail addresses of three references. (Reference letters should be provided only upon the request of the search committee).

Virginia Commonwealth University is an equal opportunity, affirmative action university providing access to education and employment without regard to age, race, color, national origin, gender, religion, sexual orientation, veteran's status, political affiliation or disability

ACM Inroads
 The magazine for computing educators worldwide
<http://inroads.acm.org>

Paving the way toward excellence in computing education

The Ultimate Online Resource for Computing Professionals & Students

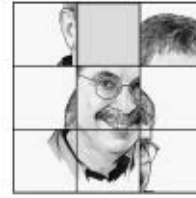
ACM DL DIGITAL LIBRARY

<http://www.acm.org/dl>



Association for Computing Machinery

Advancing Computing as a Science & Profession



DOI:10.1145/2408776.2408800

Peter Winkler

Puzzled Tumbling Dice

These puzzles involve computing probabilities associated with dice. A die is a cube with faces marked with numbers 1 to 6, as in the figure here, and we assume that when a die is rolled, each number is equally likely to come out on top. The questions we ask are somewhat unusual, though. We have collected several facts that run counter to many people's intuitions. Our job is therefore both mathematical and psychological—first, make the calculation, then, if the answer strains our intuition, try to reconcile the conflict through reasoning.

1. Six dice are rolled simultaneously, and the number N of different numbers that appear is determined; for example, if the dice show 3,4,1,6,5,6, then $N = 5$, and if they show 6,2,2,3,6,2, then $N = 3$. Clearly, N could be any number from one to six, but these values are not equally likely. What is the probability that $N = 4$?

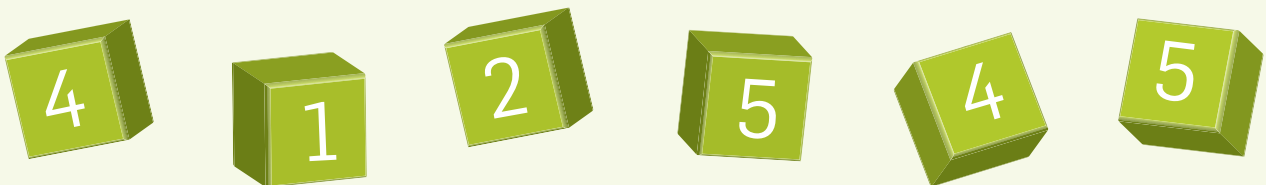
2. Alice and Bob roll a single die repeatedly. Alice is waiting until all six of the die's faces appear at least once. Bob is waiting for some face (any face) to appear four

times. The winner is the one who gets his or her wish first; for example, if the successive rolls are 2,5,4,5,3,6,6,5,1, then Alice wins, since all numbers have appeared, none more than three times. If the successive rolls instead happen to be 4,6,3,6,6,1,2,2,6, then Bob wins because he has seen four 6s and no 5 (yet). Now answer this easy question: What is the maximum number of rolls needed to determine a winner? And this more difficult question: Which player is more likely to win? This can be worked out with only a little bit of arithmetic,

assuming you are clever enough.

3. Alice and Bob have now secured a second die, roll the two dice together, noting the sum of the two values shown, and repeat. Each sum is, of course, a number from 2 to 12, with 7 the most likely outcome; six of the 36 ways to roll a pair of dice results in a sum of 7. This time, Alice is waiting for two 7s in a row, while Bob wants an 8 followed immediately by a 7. Which of them has the shorter average wait? And whose wish is more likely to come true first?

A roll of six dice from Puzzle 1 that would produce an outcome of $N = 4$.



Readers are encouraged to submit prospective puzzles for future columns to puzzled@cacm.acm.org.

Peter Winkler (puzzled@cacm.acm.org) is William Morrill Professor of Mathematics and Computer Science at Dartmouth College, Hanover, NH.

4th Annual ACM SIGPLAN Conference on
**Systems,
Programming,
Languages,
Applications:
Software for
Humanity**

Submission Deadlines

March 28, 2013

- OOPSLA Papers
- Wavefront Papers & Experience Reports
- Proposals for Workshops & Panels

April 5, 2013

- Onward! Papers & Essays

June 8, 2013

- Dynamic Languages Symposium

June 28, 2013

- Posters, Doctoral Symposium
- ACM Student Research Competition
- Demonstrations
- Student Volunteers

Location

Hyatt Regency Indianapolis

Events

- 28th Annual OOPSLA
- Onward!
- Wavefront
- Dynamic Languages Symposium (DLS)
- Generative Programming & Component Engineering (GPCE)
- Software Language Engineering (SLE)
- ...and more

General Chairs

Patrick Eugster & Antony Hosking
Purdue University

OOPSLA Papers Chair

Cristina Lopes
University of California, Irvine

Onward! Papers Chair

Robert Hirschfeld
Hasso-Plattner-Institut Potsdam

Onward! Essays Chair

Bernd Brügge
Technische Universität München

DLS Papers Chair

Carl Friedrich Bolz
Heinrich-Heine-Universität Düsseldorf

More Information
<http://splashcon.org>
info@splashcon.org



SPLASH
INDIANAPOLIS 2013
OCTOBER 26-31

