

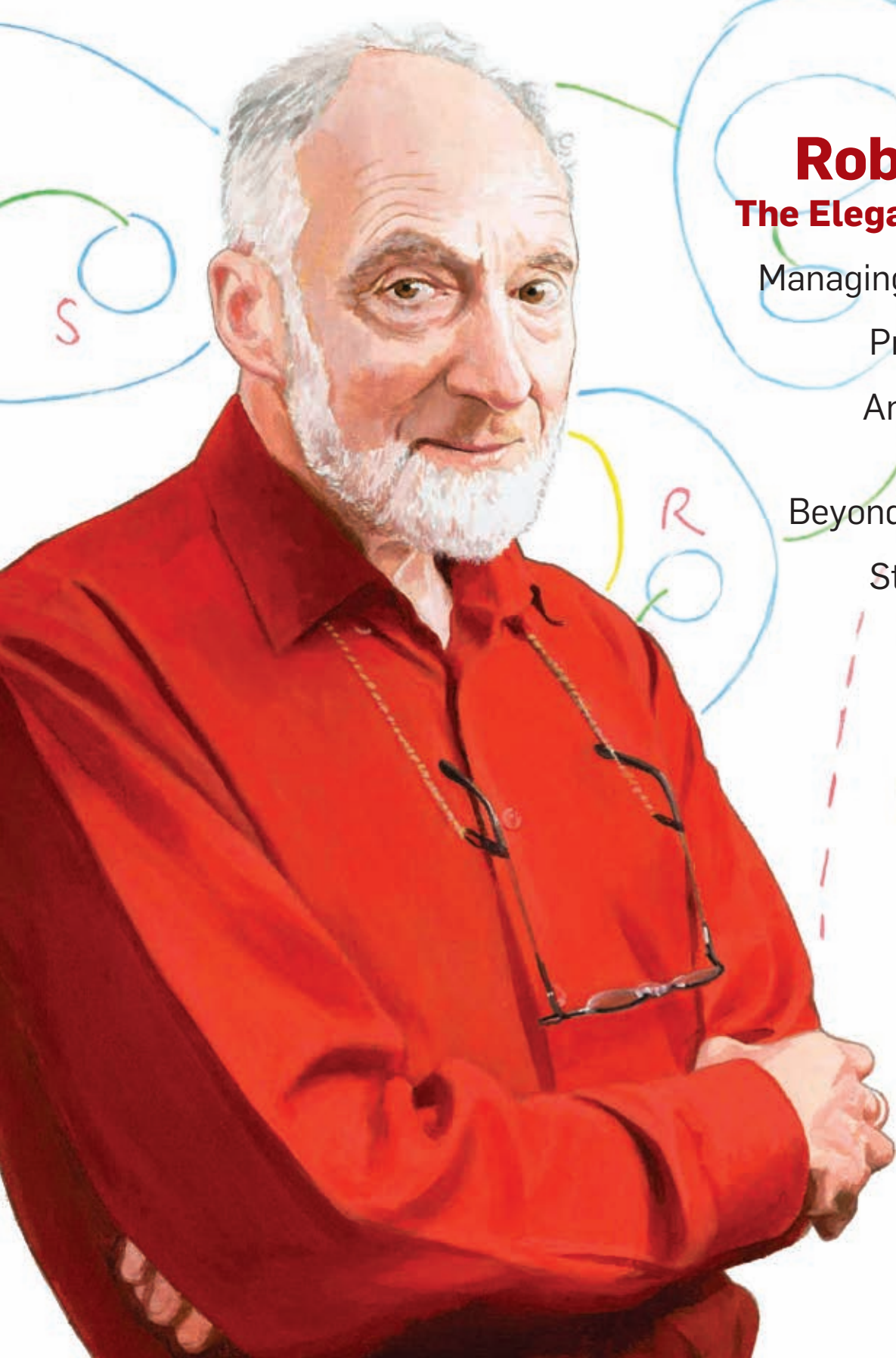
# COMMUNICATIONS

CACM.ACM.ORG

OF THE

# ACM

06/2010 VOL.53 NO.06



## **Robin Milner** **The Elegant Pragmatist**

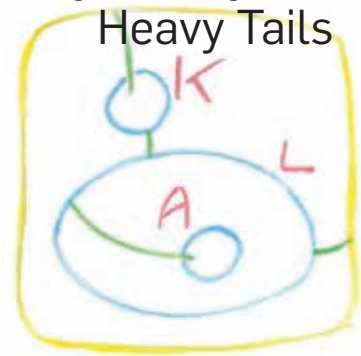
Managing Scientific Data

Privacy by Design

An Interview with  
Ed Feigenbaum

Beyond the Smart Grid

Straightening Out  
Heavy Tails



# ACM, Uniting the World's Computing Professionals, Researchers, Educators, and Students



Dear Colleague,

At a time when computing is at the center of the growing demand for technology jobs worldwide, ACM is continuing its work on initiatives to help computing professionals stay competitive in the global community. ACM's increasing involvement in activities aimed at ensuring the health of the computing discipline and profession serve to help ACM reach its full potential as a global and diverse society which continues to serve new and unique opportunities for its members.

As part of ACM's overall mission to advance computing as a science and a profession, our invaluable member benefits are designed to help you achieve success by providing you with the resources you need to advance your career and stay at the forefront of the latest technologies.

I would also like to take this opportunity to mention ACM-W, the membership group within ACM. ACM-W's purpose is to elevate the issue of gender diversity within the association and the broader computing community. You can join the ACM-W email distribution list at <http://women.acm.org/joinlist>.

## ACM MEMBER BENEFITS:

- A subscription to ACM's newly redesigned monthly magazine, **Communications of the ACM**
- Access to ACM's **Career & Job Center** offering a host of exclusive career-enhancing benefits
- **Free e-mentoring services** provided by MentorNet®
- **Full access to over 2,500 online courses** in multiple languages, and 1,000 virtual labs
- **Full access to 600 online books** from Safari® Books Online, featuring leading publishers, including O'Reilly (Professional Members only)
- **Full access to 500 online books** from Books24x7®
- Full access to the new **acmqueue** website featuring blogs, online discussions and debates, plus multimedia content
- The option to subscribe to the complete **ACM Digital Library**
- The **Guide to Computing Literature**, with over one million searchable bibliographic citations
- The option to connect with the **best thinkers in computing** by joining **34 Special Interest Groups** or **hundreds of local chapters**
- **ACM's 40+ journals and magazines** at special member-only rates
- **TechNews**, ACM's tri-weekly email digest delivering stories on the latest IT news
- **CareerNews**, ACM's bi-monthly email digest providing career-related topics
- **MemberNet**, ACM's e-newsletter, covering ACM people and activities
- **Email forwarding service & filtering service**, providing members with a free acm.org email address and **Postini** spam filtering
- And much, much more

ACM's worldwide network of over 92,000 members range from students to seasoned professionals and includes many of the leaders in the field. ACM members get access to this network and the advantages that come from their expertise to keep you at the forefront of the technology world.

Please take a moment to consider the value of an ACM membership for your career and your future in the dynamic computing profession.

Sincerely,

Wendy Hall

A blue ink handwritten signature of Wendy Hall, written in a cursive style.

President  
Association for Computing Machinery



Association for  
Computing Machinery

*Advancing Computing as a Science & Profession*



Association for  
Computing Machinery

Advancing Computing as a Science & Profession

# membership application & digital library order form

Priority Code: ACACM10

## You can join ACM in several easy ways:

<b>Online</b> <a href="http://www.acm.org/join">http://www.acm.org/join</a>	<b>Phone</b> +1-800-342-6626 (US & Canada) +1-212-626-0500 (Global)	<b>Fax</b> +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/serving/ethics.html>

## 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](http://www.acm.org)

Professional membership dues include \$40 toward a subscription to *Communications of the ACM*. 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](mailto: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 \_\_\_\_\_

## Departments

- 5 **ACM's Chief Operating Officer Letter  
A Tour of ACM's HQ**  
*By Patricia Ryan*
- 
- 6 **Letters to the Editor  
Workflow Tools for  
Distributed Teams?**
- 
- 8 **In the Virtual Extension**
- 
- 10 **BLOG@CACM  
The Chaos of the Internet  
as an External Brain; and More**  
Greg Linden writes about the Internet as a peripheral resource; Ed H. Chi discusses lessons learned from the DARPA Network Challenge; and Mark Guzdial asks if there are too many IT workers or too many IT jobs.
- 
- 12 **CACM Online  
Interact Naturally**  
*By David Roman*
- 
- 29 **Calendar**
- 
- 116 **Careers**

## Last Byte

- 118 **Puzzled  
Solutions and Sources**  
*By Peter Winkler*
- 
- 120 **Future Tense  
How the Net Ensures  
Our Cosmic Survival**  
Give adolescent wonder an evolutionary jolt.  
*David Brin*

## News

- 13 **Straightening Out Heavy Tails**  
A better understanding of heavy-tailed probability distributions can improve activities from Internet commerce to the design of server farms.  
*By Neil Savage*
- 
- 16 **Beyond the Smart Grid**  
Sensor networks monitor residential and institutional devices, motivating energy conservation.  
*By Tom Geller*
- 
- 18 **Mine Your Business**  
Researchers are developing new techniques to gauge employee productivity from information flow.  
*By Leah Hoffmann*
- 
- 20 **Robin Milner:  
The Elegant Pragmatist**  
Remembering a rich legacy in verification, languages, and concurrency.  
*By Leah Hoffmann*
- 
- 22 **CS and Technology Leaders Honored**  
*By Jack Rosenberger*



inspiration for this month's cover. For more on Jammet, see <http://www.heartagency.com/artist/JosieJammet/>

**About the Cover:**  
"Dreams of perfection, for a scientist, mean avenues for exploration," said Robin Milner in an interview shortly before his death on March 20, 2010 at age 76. Photographer Roland Eva captured Milner, a renowned CS theorist and recipient of the 1991 ACM A.M. Turing Award, in one of his favorite places—the classroom. Josie Jammet used that picture as

## Viewpoints

- 24 **Privacy and Security  
Myths and Fallacies of "Personally  
Identifiable Information"**  
Developing effective privacy protection technologies is a critical challenge.  
*By Arvind Narayanan  
and Vitaly Shmatikov*
- 
- 27 **Inside Risks  
Privacy By Design: Moving  
from Art to Practice**  
Designing privacy into systems at the beginning of the development process.  
*By Stuart S. Shapiro*
- 
- 30 **The Profession of IT  
The Resurgence of Parallelism**  
Parallel computation is making a comeback after a quarter century of neglect.  
*By Peter J. Denning and Jack B. Dennis*
- 
- 33 **Kode Vicious  
Plotting Away**  
Tips and tricks for visualizing large data sets.  
*By George V. Neville-Neil*
- 
- 35 **Law and Technology  
Intel's Rebates: Above Board  
or Below the Belt?**  
Over several years, Intel paid billions of dollars to its customers. Was it to force them to boycott products developed by its rival AMD or so they could sell its microprocessors at lower prices?  
*By François Lévêque*
- 
- 38 **Viewpoint  
Institutional Review Boards  
and Your Research**  
A proposal for improving the review procedures for research projects that involve human subjects.  
*By Simson L. Garfinkel  
and Lorrie Faith Cranor*
- 
- 41 **Interview  
An Interview with Ed Feigenbaum**  
*By Len Shustek*



## Practice



- 46 **Securing Elasticity in the Cloud**  
Elastic computing has great potential, but many security challenges remain.  
*By Dustin Owens*
- 52 **Simplicity Betrayed**  
Emulating a video system shows how even a simple interface can be more complex—and capable—than it appears.  
*By George Phillips*
- 59 **A Tour Through the Visualization Zoo**  
A survey of powerful visualization techniques, from the obvious to the obscure.  
*By Jeffrey Heer, Michael Bostock, and Vadim Ogievetsky*

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

## Contributed Articles

- 68 **Managing Scientific Data**  
Needed are generic, rather than one-off, DBMS solutions automating storage and analysis of data from scientific collaborations.  
*By Anastasia Ailamaki, Verena Kantere, and Debabrata Dash*
- 79 **Conference Paper Selectivity and Impact**  
Conference acceptance rate signals future impact of published conference papers.  
*By Jilin Chen and Joseph A. Konstan*

## Review Articles

- 84 **Efficiently Searching for Similar Images**  
New algorithms provide the ability for robust but scalable image search.  
*By Kristen Grauman*

## Research Highlights

- 96 **Technical Perspective**  
**Building Confidence in Multicore Software**  
*By Vivek Sarkar*
- 97 **Asserting and Checking Determinism for Multithreaded Programs**  
*By Jacob Burnim and Koushik Sen*
- 106 **Technical Perspective**  
**Learning To Do Program Verification**  
*By K. Rustan M. Leino*
- 107 **seL4: Formal Verification of an Operating-System Kernel**  
*By Gerwin Klein, June Andronick, Kevin Elphinstone, Gernot Heiser, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood*

## Virtual Extension

As with all magazines, page limitations often prevent the publication of articles that might otherwise be included in the print edition. To ensure timely publication, ACM created *Communications'* Virtual Extension (VE).

VE articles undergo the same rigorous review process as those in the print edition and are accepted for publication on their merit. These articles are now available to ACM members in the Digital Library.

**Examining Agility in Software Development Practice**

*Sergio de Cesare, Mark Lycett, Robert D. Macredie, Chaitali Patel, and Ray Paul*

**Barriers to Systematic Model Transformation Testing**

*Benoit Baudry, Sudipto Ghosh, Franck Fleurey, Robert France, Yves Le Traon, and Jean-Marie Mottu*

**Factors that Influence Software Piracy:**

**A View from Germany**  
*Alexander Nill, John Schibrowsky, James W. Peltier, and Irvin L. Young*

**The Requisite Variety of Skills for IT Professionals**

*Kevin P. Gallagher, Kate M. Kaiser, Judith C. Simon, Cynthia M. Beath, and Tim Goles*

**Panopticon Revisited**

*Jan Kietzmann and Ian Angell*

**The Social Influence Model of Technology Adoption**

*Sandra A. Vannoy and Prashant Palvia*

**I, Myself and e-Myself**

*Cheul Rhee, G. Lawrence Sanders, and Natalie C. Simpson*

**Beyond Connection: Situated Wireless Communities**

*Jun Sun and Marshall Scott Poole*



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 Membership**  
 Lillian Israel

**Director, Office of SIG Services**  
 Donna Cappel

**Director, Office of Publications**  
 Bernard Rous

**Director, Office of Group Publishing**  
 Scott Delman

**ACM COUNCIL**

**President**

Wendy Hall

**Vice-President**

Alain Chesnais

**Secretary/Treasurer**

Barbara Ryder

**Past President**

Stuart I. Feldman

**Chair, SGB Board**

Alexander Wolf

**Co-Chairs, Publications Board**

Ronald Boisvert, Holly Rushmeier

**Members-at-Large**

Carlo Ghezzi;

Anthony Joseph;

Mathai Joseph;

Kelly Lyons;

Bruce Maggs;

Mary Lou Soffa;

Fei-Yue Wang

**SGB Council Representatives**

Joseph A. Konstan;

Robert A. Walker;

Jack Davidson

**PUBLICATIONS BOARD**

**Co-Chairs**

Ronald F. Boisvert and Holly Rushmeier

**Board Members**

Jack Davidson; Nikil Dutt; Carol Hutchins;

Ee-Peng Lim; Catherine McGeoch;

M. Tamer Ozsu; Vincent Shen;

Mary Lou Soffa; Ricardo Baeza-Yates

**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

2 Penn Plaza, Suite 701

New York, NY 10121-0701 USA

T (800) 401-1799; F (541) 687-1840

**Association for Computing Machinery (ACM)**

2 Penn Plaza, Suite 701

New York, NY 10121-0701 USA

T (212) 869-7440; F (212) 869-0481

# 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**

Alicia Kubista

**Assistant Art Director**

Mia Angelica Balaquiot

**Production Manager**

Lynn D'Addesio

**Director of Media Sales**

Jennifer Ruzicka

**Marketing & Communications Manager**

Brian Hebert

**Public Relations Coordinator**

Virginia Gold

**Publications Assistant**

Emily Eng

**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**

acmcoa@cacm.acm.org

**Letters to the Editor**

letters@cacm.acm.org

**WEB SITE**

http://cacm.acm.org

**AUTHOR GUIDELINES**

http://cacm.acm.org/guidelines

**ADVERTISING**

**ACM ADVERTISING DEPARTMENT**

2 Penn Plaza, Suite 701, New York, NY

10121-0701

T (212) 869-7440

F (212) 869-0481

**Director of Media Sales**

Jennifer Ruzicka

jen.ruzicka@hq.acm.org

**Media Kit** acmm mediasales@acm.org

**EDITORIAL BOARD**

**EDITOR-IN-CHIEF**

Moshe Y. Vardi  
 eic@cacm.acm.org

**NEWS**

**Co-chairs**

Marc Najork and Prabhakar Raghavan

**Board Members**

Brian Bershad; Hsiao-Wuen Hon;

Mei Kobayashi; Rajeev Rastogi;

Jeannette Wing

**VIEWPOINTS**

**Co-chairs**

Susanne E. Hambrusch; John Leslie King;

J Strother Moore

**Board Members**

P. Anandan; William Aspray;

Stefan Bechtold; Judith Bishop;

Stuart I. Feldman; Peter Freeman;

Seymour Goodman; Shane Greenstein;

Mark Guzdial; Richard Heeks;

Rachelle Hollander; Richard Ladner;

Susan Landau; Carlos Jose Pereira de Lucena;

Beng Chin Ooi; Loren Terveen

**Q PRACTICE**

**Chair**

Stephen Bourne

**Board Members**

Eric Allman; Charles Beeler; David J. Brown;

Bryan Cantrill; Terry Coatta; Mark Compton;

Stuart Feldman; Benjamin Fried;

Pat Hanrahan; Marshall Kirk McKusick;

George Neville-Neil; Theo Schlossnagle;

Jim Waldo

The Practice section of the CACM

Editorial Board also serves as

the Editorial Board of *ACM Queue*.

**CONTRIBUTED ARTICLES**

**Co-chairs**

Al Aho and Georg Gottlob

**Board Members**

Yannis Bakos; Gilles Brassard; Alan Bundy;

Peter Buneman; Ghezzi Carlo;

Andrew Chien; Anja Feldmann;

Blake Ives; James Larus; Igor Markov;

Gail C. Murphy; Shree Nayar; Lionel M. Ni;

Sriram Rajamani; Jennifer Rexford;

Marie-Christine Rousset; Avi Rubin;

Abigail Sellen; Ron Shamir; Marc Snir;

Larry Snyder; Veda Storey;

Manuela Veloso; Michael Vitale;

Wolfgang Wahlster; Andy Chi-Chih Yao;

Willy Zwaenepoel

**RESEARCH HIGHLIGHTS**

**Co-chairs**

David A. Patterson and Stuart J. Russell

**Board Members**

Martin Abadi; Stuart K. Card; Deborah Estrin;

Shafi Goldwasser; Monika Henzinger;

Maurice Herlihy; Norm Jouppi;

Andrew B. Kahng; Gregory Morrisett;

Michael Reiter; Mendel Rosenblum;

Ronitt Rubinfeld; David Salesin;

Lawrence K. Saul; Guy Steele, Jr.;

Gerhard Weikum; Alexander L. Wolf;

Margaret H. Wright

**WEB**

**Co-chairs**

Marti Hearst and James Landay

**Board Members**

Jason I. Hong; Jeff Johnson;

Greg Linden; Wendy E. MacKay



**ACM Copyright Notice**

Copyright © 2010 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) 626-0654.

**Single Copies**

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

**COMMUNICATIONS OF THE ACM**

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

**POSTMASTER**

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



Association for Computing Machinery



Printed in the U.S.A.

DOI:10.1145/1743546.1743547

Patricia Ryan

# A Tour of ACM's HQ

Let me admit at the outset, I'm a bit biased when I talk about ACM headquarters (HQ) and the many ways it outshines the command centers of other professional

societies. But that bias comes from nearly three decades of firsthand experience, having worked in almost every department within HQ, and having had a front-row seat to the dynamics that drive each department to create and provide products and services to meet the needs of ACM members.

The HQ staff coordinates the global activities of ACM's chapters and various committees; acting as the liaison for all conferences carrying the ACM stamp; enhancing the Digital Library (DL); producing almost four dozen publications; providing professional services and online courses; and performing organizational functions. HQ also serves as the hub for members, news media, and the public on all subjects related to computing and technologies.

Probably the greatest difference between ACM and other professional societies is just how much is accomplished by so few. With a membership fast approaching 100,000, ACM's 170 conferences, 45 periodicals, 34 Special Interest Groups, and 644 professional and student chapters are all supported by 72 staffers. When you consider that similar professional associations with equal or fewer members and services are often supported by staffs of over 100, I hope you are as impressed, as I continue to be, by the devotion of the people working at ACM.

Here, I'd like to briefly highlight the departments and talented directors that spearhead ACM's New York HQ.

ACM's Special Interest Groups (SIGs) represent virtually every major discipline within computer science.

ACM and its SIGs sponsor, co-sponsor, and cooperate with over 170 technical meetings annually, all of which are coordinated through the **Office of SIG Services**, with Donna Cappo at the helm. The department manages all SIGs and their conferences, including promotion, publicity, and membership. The team also helps produce over 85 proceedings and 30 newsletters yearly.

The **Office of Publications** produces ACM's many research journals and *Transactions* and is responsible for Digital Library content. Director Bernard Rous oversees these efforts, which include 39 print journals and a DL containing over 275,000 articles and 1.50 million citations covered by the *Guide to Computing Literature*.

Scott Delman, director of the **Office of Group Publishing**, shepherds DL sales and builds business strategies to make the DL a must-have subscription for institutions worldwide, which today number over 2,800. He also serves as group publisher for ACM's magazine series, including our flagship *Communications of the ACM*.


ACM's **Office of Membership**, under the leadership of Lillian Israel, is one big umbrella covering everything under membership and marketing services, including membership processing, subscriptions, professional development services, and education. Of particular pride to me is the fact that with close to 100,000 members, ACM's member services department, consisting of *four* customer representatives, is able to provide a response to any member query within 24 hours!

Russell Harris, a 36-year veteran of HQ, heads the **Office of Financial Services**. His team oversees all the Association's financial matters, including accounting responsibilities, monthly and quarterly financial statements, and the annual ACM budget. His department cuts 10,000 checks per year, guides auditors through evaluations, handles treasury functions, and monitors ACM's investment strategies daily.

The **Office of Information Systems** maintains constant vigilance over the Association's "computing machinery," with responsibilities for all the information processing needs serving staffers, volunteers, and members. Wayne Graves leads this effort, which includes maintaining HQ's digital infrastructure, as well as spearheading the DL's technical direction and advances.

Based in Washington, D.C., ACM's **Office of Public Policy** (USACM) is very much a part of HQ, representing ACM's interests on IT policy issues that impact the computing field. Cameron Wilson works to enlighten legislators about policies that foster computing innovations, including the critical need for CS and STEM education.

The unsung heroes that comprise the **Office of Administrative Services** support all the necessary functions that allow HQ to operate at the top of its game every single day. These varied services include human resources, office services and mailroom facilities, and support policies and procedures, the ACM's Awards program, and elections.

All of us at HQ are indebted to the tireless efforts of ACM's many devoted volunteers worldwide who work together with us to manage ACM's growing array of products and services. None of our efforts would succeed without their efforts. 

Patricia Ryan is Deputy Executive Director and Chief Operating Officer of ACM, New York, NY.

© 2010 ACM 0001-0782/10/0600 \$10.00

# Workflow Tools for Distributed Teams?

**T**HE “PROFESSION OF IT” Viewpoint “Orchestrating Coordination in Pluralistic Networks” by Peter J. Denning et al. (Mar. 2010) offered guidance for distributed development teams. As a leader of one such team, I can vouch for the issues it raised. However, my coordination problems are compounded because email (and related attachments) is today’s de facto medium for business and technical communication. The most up-to-date version of a document is an email attachment that instantly goes out of date when changes are made by any of the team members; project documents include specifications, plans, status reports, assignments, and schedules.

Software developers use distributed source-code control systems to manage changes to code. But these tools don’t translate well to all the documents handled by nondevelopers, including managers, marketers, manufacturers, and service and support people. I’d like to know what workflow-management tools Denning et al. would recommend for such an environment.

**Ronnie Ward**, Houston, TX

## Author’s Response:

*Workflow tools are not the issue. Many people simply lack a clear model of coordination. They think coordination is about exchanging messages and that related coordination breakdowns indicate poorly composed, garbled, or lost messages (as in email). Coordination is about making commitments, usually expressed as “speech acts,” or utterances that take action and make the commitments that produce the outcome the parties want. People learning the basics of coordination are well on their way toward successful coordination, even without workflow tools.*

*We don’t yet know enough about effective practices for pluralistic coordination to be able to design good workflow tools for this environment.*

**Peter J. Denning**, Monterey, CA

## Time to Debug

George V. Neville-Neil’s “Kode Vicious” Viewpoint “Taking Your Network’s Temperature” (Feb. 2010) was thought-provoking, but two of its conclusions—“putting `printf()`... throughout your code is a really annoying way to find bugs” and “limiting the files to one megabyte is a good start”—were somewhat misleading.

Timing was one reason Neville-Neil offered for his view that `printf()` can lead to “erroneous results.” Debugger and `printf()` both have timing loads. Debug timing depends on hardware support. A watch statement functions like a `printf()`, and a breakpoint consumes “infinite” time. In both single-threaded and multithreaded environments, a breakpoint stops thread activity. In all cases, debugger statements perturb timing in a way that’s like `printf()`.

We would expect such stimulus added to multithreaded applications would produce different output. Neville-Neil expressed a similar sentiment, saying “Networks are perhaps the most nondeterministic components of any complex computing system.” Both `printf()` and debuggers exaggerate timing differences, so the qualitative issue resolves to individual preferences, not to timing.

Choosing between a debugger and a `printf()` statement depends on the development stage in which each is to be used. At an early stage, a debugger might be better when timing and messaging order are less important than error detection. Along with functional integration in the program, a debugger can sometimes reach a point of diminishing returns. Programmers shift their attention to finding the first appearance of an error and the point in their programs where the error was generated. Using a debugger tends to be a trial-and-error process involving large amounts of programmer and test-bench time to find that very point. A `printf()` statement inserted at program creation requires no setup

time and little bench time, so is, in this sense, resource-efficient.

The downside of using a `printf()` statement is that at program creation (when it is inserted) programmers anticipate errors but are unaware of where and when they might occur; `printf()` output can be overwhelming, and the aggregate time to produce diagnostic output can impede time-critical operations. The overhead load of output and time is only partially correctable.

Limiting file size to some arbitrary maximum leads programmers to assume (incorrectly) that the search is for a single error and that localizing it is the goal. Limiting file size allows programmers to focus on a manageable subset of data for analysis but misses other unrelated errors. If the point of error-generation is not within some limited number of files, little insight would be gained for finding the point an error was in fact generated.

Neville-Neil saying “No matter how good a tool you have, it’s going to do a much better job at finding a bug if you narrow down the search.” might apply to “Dumped” (the “questioner” in his Viewpoint) but not necessarily to everyone else. An analysis tool is meant to discover errors, and programmers and users both win if errors are found. Trying to optimize tool execution time over error-detection is a mistake.

**Art Schwarz**, Irvine, CA

George V. Neville-Neil’s Viewpoint (Feb. 2010) said students are rarely taught to use tools to analyze networking problems. For example, he mentioned Wireshark and `tcpdump`, but only in a cursory way, even though these tools are part of many contemporary university courses on networking.

Sniffers (such as Wireshark and Ethereal) for analyzing network protocols have been covered at Fairleigh Dickinson University for at least the past 10 years. Widely used tools for network analysis and vulnerability assessment (such as `nmap`, `nessus`, `Snort`, and `ettercap`) are available through Fedora and nUbuntu Linux



distributions. Open source tools for wireless systems include NetStumbler and AirSnort.

Fairleigh Dickenson's network labs run on virtual machines to limit inadvertent damage and the need for protection measures. We teach the basic network utilities available on Windows- and/or Posix-compliant systems, including ping, netstat, arp, tracer (tracert), ipconfig (ifconfig in Linux/Unix and iwconfig in Linux wireless cards), and nslookup (dig in Linux). With the proper options, netstat displays IP addresses, protocols, and ports used by all open and listening connections, as well as by protocol statistics and routing tables.

The Wireshark packet sniffer identifies control information at different protocol layers. A TCP capture specification thus provides a tree of protocols, with fields for frame header and trailer (including MAC address), IP header (including IP address), and TCP header (including port address). Students compare the MAC and IP addresses found through Wireshark with those found through netstat and ipconfig. They then change addresses and check results by sniffing new packets, analyzing the altered addresses. Capture filters in Wireshark support search through protocol and name resolution; Neville-Neil stressed the importance of narrowing one's search but failed to mention the related mechanisms. Students are also able to make connections through (unencrypted) telnet and PuTTY, comparing password fields.

My favorite Wireshark assignment involves viewing TCP handshakes via statistics/flow/TCP flow, perhaps following an nmap SYN attack. The free security scanner nmap runs with Wireshark and watch probes initiated by the scan options provided. I always assign a Christmas-tree scan (nmap -sX) that sends packets with different combinations of flag bits. Capturing probe packets and a receiving station's reactions enables identification of flag settings and the receiver's response to them. Operating systems react differently to illegal flag combinations, as students observe via their screen captures.

Network courses and network maintenance are thus strongly enhanced by sniffers and other types of tools that yield

information concerning network traffic and potential system vulnerabilities.

**Gertrude Levine**, Madison, NJ

### What Jack Doesn't Know About Software Maintenance

I agree that the community doesn't understand software maintenance, as covered in the article "You Don't Know Jack about Software Maintenance" by Paul Stachour and David Collier-Brown (Nov. 2009), but much more can be done to improve the general understanding of the important challenges.

The software-maintenance projects I've worked on have been difficult, due to the fact that maintenance work is so different from the kind of work described in the article. The community does not fully understand that maintenance involves much more than just adding capabilities and fixing bugs. For instance, maintenance teams on large projects spend almost as much time providing facility, operations, product, and sustaining-engineering support as they do changing code.<sup>1</sup> Moreover, the work tends to be distributed differently. My colleagues and I recently found maintenance teams spending as much as 60% of their effort testing code once the related changes are implemented.

Other misconceptions include:

*The primary job in maintenance is facilitating changes.* We found that support consumes almost as much effort as changes and repairs;

*Maintenance is aimed at addressing new requirements.* Because most jobs are small, maintenance teams focus on closing high-priority trouble reports rather than making changes;

*Funding maintenance is based on requirements.* Most maintenance projects are funded level-of-effort; as such, maintenance managers must determine what they can do with the resources they have rather than what needs to be done;

*Maintenance schedules are based on user-need dates.* Maintenance schedules are written in sand, so maintenance leaders must determine what can be done within a limited time period;

*Maintenance staff is junior.* Average experience for maintenance personnel is 25 years during which they tend to

work on outdated equipment to fix software written in aging languages; and

*Maintenance is well tooled.* We found the opposite. Maintenance tools are inferior, and development tools and regression test suites do not unfortunately support the work.

Maintenance involves much more than Stachour and Collier-Brown indicated. In light of the changing nature of the work being done every day by software maintenance teams, my colleagues and I urge *Communications* to continue to cover the topic.

#### Reference

1. Reifer, D. Allen, J.-A., Fersch, B., Hitchings, B., Judy, J., and Rosa, W. Software maintenance: Debunking the myths. In *Proceedings of the International Society of Parametric Analysis / Society of Cost Estimating and Analysis Annual Conference and Training Workshop* (San Diego, CA, June 8-11). ISPA/SCEA, Vienna, VA, 2010.

**Donald J. Reifer**, Prescott, AZ

### Authors' Response:

*In our slightly tongue-in-cheek description of software maintenance, we were concentrating on the "add a highway" side of the overall problem, rather than "repair the railroad bridge." We try to avoid considering software maintenance as a separate process done by a different team. That's a genuinely difficult problem, as Reifer points out. We've seen it tried a number of times, with generally disappointing results.*

*A better question might be the one asked by Drew Sullivan, president of the Greater Toronto Area Linux User Group, at a presentation we gave on the subject: "Why aren't you considering maintenance as continuing development?" In fact we were, describing the earlier Multics norm of continuous maintenance without stopping any running programs. We're pleased to see the continuous process being independently reinvented by practitioners of the various agile methods. In addition, we're impressed by their refactoring and test-directed development. These are genuinely worthwhile improvements to the continuous approach, and we hope the techniques we re-described are valuable to that community.*

**Paul Stachour**, Bloomington, MN

**David Collier-Brown**, Toronto

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

© 2010 ACM 0001-0782/10/0600 \$10.00

# In the Virtual Extension

Communications' *Virtual Extension* brings more quality articles to ACM members. These articles are now available in the ACM Digital Library.

## Examining Agility in Software Development Practice

*Sergio de Cesare, Mark Lycett, Robert D. Macredie, Chaitali Patel, and Ray Paul*

Agility is a facet of software development attracting increasing interest. The authors investigate the value of agility in practice and its effects on traditional plan-based approaches. Data collected from senior development/project managers in 62 organizations is used to investigate perceptions related to agile development. Specifically, the perceptions tested relate to the belief in agile values and principles, and the value of agile principles within current development/organization practice. These perceptions are examined in the context of current practice in order to test perceptions against behavior and understand the valued aspects of agile practice implicit in development today. The broad outcome indicates an interesting marriage between agile and plan-based approaches. This marriage seeks to allow flexibility of method while retaining control.

## Barriers to Systematic Model Transformation Testing

*Benoit Baudry, Sudipto Ghosh, Franck Fleurey, Robert France, Yves Le Traon, and Jean-Marie Mottu*

Model Driven Engineering (MDE) techniques support extensive use of models in order to manage the increasing complexity of software systems. Automatic model transformations play a critical role in MDE since they automate complex, tedious, error-prone, and recurrent software development tasks. For example, Airbus uses automatic code synthesis from SCAD models to generate the code for embedded controllers in the Airbus A380. Model transformations that automate critical software development tasks must be validated. The authors identify characteristics of model transformation approaches that contribute to the difficulty of systematically testing transformations as well as present promising solutions and propose possible ways to overcome these barriers.

## Factors that Influence Software Piracy: A View from Germany

*Alexander Nill, John Schibrowsky, James W. Peltier, and Irvin L. Young*

Software piracy has wide-ranging negative economic consequences for manufacturers and distributors striving to compete in a competitive global market. Indeed, software piracy is jeopardizing the future growth and development of the IT industry, which in turn disproportionately impacts countries with the highest piracy rates. This article details an exploratory study that investigated the relationship between a comprehensive set of factors and software piracy in Germany. The authors gleaned some valuable security measures from the results of the study that can be used as a starting point for industry and governments to develop programs to deter piracy.

## The Requisite Variety of Skills for IT Professionals

*Kevin P. Gallagher, Kate M. Kaiser, Judith C. Simon, Cynthia M. Beath, and Tim Goles*

IT professionals today are beset by ongoing changes in technology and business practices. To thrive in such a dynamic environment requires competency in a broad range of skills, both technical and nontechnical. The authors contend the Law of Requisite Variety—adapting to change requires a varied enough solution set to match the complexity of the environment—can help explain the need for greater and broader skills among IT professionals. The article outlines a framework containing six skill categories critically important for the career development of IT professionals.

## Panopticon Revisited

*Jan Kietzmann and Ian Angell*

Many claims have been made regarding the safety benefits of computer-supported surveillance technologies. However, like many technologies the advantageous door swings both ways. The authors compare how current computer and communication technologies are shaping today's "panopticons," pulling heavily from the 1787 prison architectural design by social theorist Jeremy Bentham that allowed prison officials and observers to keep an eye on prisoners without the imprisoned able to tell they are being watched.

## The Social Influence Model of Technology Adoption

*Sandra A. Vannoy and Prashant Palvia*

While social computing has fast become an industry buzzword encompassing networking, human innovation, and communications technologies, few studies have investigated technology adoption targeting the individual at the level of society, community, or lifestyle experience. The authors address this gap by developing social constructs and providing a theoretically grounded model for technology adoption in the context of social computing. Their model suggests that social computing action, cooperation, consensus, and authority are antecedents to social influence. And social influence, they contend, leads to technology adoption.

## I, Myself and e-Myself

*Cheul Rhee, G. Lawrence Sanders, and Natalie C. Simpson*

The human ego, developed from birth, is central to one's conscious self, according to experts. This article examines the concept of the *virtual* ego (one that begins with the creation of an online identity and functions only online) and the notion of an online persona as overarching concepts providing a new rationale for understanding and explaining online behavior. The authors posit that an Internet user's virtual ego is distinguishable from his or her real ego and argue that understanding the differences between the two is essential for advancing the dynamics of Web-based social networks.

## Beyond Connection: Situated Wireless Communities

*Jun Sun and Marshall Scott Poole*

Compared to traditional Internet-based virtual communities, situated wireless communities (SWCs) go beyond just connecting people together. In fact, SWCs enable people to share their common physical and/or social context with each other. With the availability of these cues, the social interaction among members within a community is significantly enhanced. The authors detail four general types of SWCs as well as give examples of the strengths and weaknesses of each.



# THE ACM A. M. TURING AWARD



ACM, INTEL, AND GOOGLE  
CONGRATULATE  
**CHARLES P. THACKER**  
FOR THE PIONEERING  
DESIGN AND REALIZATION  
OF THE FIRST MODERN  
PERSONAL COMPUTER—  
THE ALTO AT XEROX PARC—  
AND SEMINAL INVENTIONS  
AND CONTRIBUTIONS TO  
LOCAL AREA NETWORKS  
(INCLUDING THE ETHERNET),  
MULTIPROCESSOR  
WORKSTATIONS, SNOOPING  
CACHE COHERENCE  
PROTOCOLS, AND TABLET  
PERSONAL COMPUTERS

BY THE COMMUNITY...

FROM THE COMMUNITY...

FOR THE COMMUNITY...



“ Charles Thacker’s Alto computer embodied the key elements of today’s PCs, and is at the root of one of the world’s most innovative industries. Intel applauds Chuck’s clarity of insight, focus on simplicity, and amazing track-record of designing landmark systems that have accelerated the progress of research and industry for decades.”

**Andrew A. Chien**  
Vice President, Research  
Director, Future Technologies Research

For more information see [www.intel.com/research](http://www.intel.com/research).



“ Google is pleased to honor contributions that made possible the style of computing that we enjoy today. We are proud to support the Turing Award to encourage continued research in computer science and the related technologies that depend on its continued advancement.”

**Alfred Spector**  
Vice President, Research and  
Special Initiatives, Google

For more information, see <http://www.google.com/corporate/index.html> and <http://research.google.com/>.



Financial support for the ACM A. M. Turing Award is provided by Intel Corporation and Google.



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.



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

DOI:10.1145/1743546.1743551

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

## The Chaos of the Internet as an External Brain; and More

*Greg Linden writes about the Internet as a peripheral resource; Ed H. Chi discusses lessons learned from the DARPA Network Challenge; and Mark Guzdial asks if there are too many IT workers or too many IT jobs.*



### Greg Linden's "The Rise of the External Brain"

<http://cacm.acm.org/blogs/blog-cacm/54333>

From the early days of computers, people have speculated that computers would be used to supplement our intelligence. Extended stores of knowledge, memories once forgotten, computational feats, and expert advice would all be at our fingertips.

In the last decades, most of the work toward this dream has been in the form of trying to build artificial intelligence. By carefully encoding expert knowledge into a refined and well-pruned database, researchers strove to build a reliable assistant to help with tasks. Sadly, this effort was always thwarted by the complexity of the system and environment, with too many variables and uncertainty for any small team to fully anticipate.

Success now is coming from an entirely unexpected source—the chaos of

the Internet. Google has become our external brain, sifting through the extended stores of knowledge offered by multitudes, helping us remember what we once found, and locating advice from people who have been where we now want to go.

For example, the other day I was trying to describe to someone how mitochondria oddly have a separate genome, but could not recall the details. A search for "mitochondria" yielded a Wikipedia page that refreshed my memory. Later, I was wondering if traveling by train or flying between Venice and Rome was a better choice; advice arrived immediately on a search for "train flying venice rome." Recently, I had forgotten the background of a colleague, which was restored again with a quick search on her name. Hundreds of times a day, I access this external brain, supplementing what is lost or incomplete in my own.

This external brain is not programmed with knowledge, at least

not in the sense we expected. There is no system of rules, no encoding of experts, no logical reasoning. There is precious little understanding of information, at least not in the search itself. There is knowledge in the many voices that make up the data on the Web, but no synthesis of those voices.

Perhaps we should have expected this. Our brains, after all, are a controlled storm of competing patterns and signals, a mishmash of evolutionary agglomeration that is barely functional and easily fooled. From this chaos can come brilliance, but also superstition, illusion, and psychosis. While early studies of the brain envisioned it as a disciplined and orderly structure, deeper investigation has proved otherwise.

And so it is fitting that the biggest progress on building an external brain also comes from chaos. Search engines pick out the gems in a democratic sea of competing signals, helping us find the brilliance that we seek. Occasionally, our external brain leads us astray, as does our internal brain, but therein lies both the risk and beauty of building a brain on disorder.



### Ed H. Chi's "The DARPA Network Challenge and the Design of Social Participation Systems"

<http://cacm.acm.org/blogs/blog-cacm/60832>

The DARPA Network Challenge recently made quite a splash across the Internet and the media. The task was to



identify the exact location of 10 red weather balloons around the country. The winning team from MIT succeeded in identifying the locations of the balloons in less than nine hours.

There was recently a good article in *Scientific American* about the winning entry and the second-place team from Georgia Tech. The article details the way in which the teams tried to: (1) build social incentives into the system to get people to participate and to recommend their friends to participate; (2) how they managed to fight spam or noisy information from people trying to lead them astray. The MIT team, for example, required photo proofs of both the balloon and the official DARPA certificate of the balloon at each location, suggesting they realized that noisy or bad data is a real challenge in social participation systems.

But what did the challenge really teach us?

Looking back for the last decade or so, we have now gotten a taste of how mass-scale participation in social computing systems results in dramatic changes in the way science, government, health care, entertainment, and enterprises operate.

The primary issue relating to the design of social participation systems is understanding the relationship between usability, sociability, social capital, collective intelligence, and how to elicit effective action through design.

► *Usability* concerns the ability for all users to contribute, regardless of their accessibility requirements and computing experience, and how to lower the interaction costs of working with social systems.

► *Sociability* refers to the skill or tendency of being sociable and of interacting well with others. There is a huge role in how the designer can facilitate and lubricate social interactions amongst users of a system.

► *Social Capital* refers to positions that people occupy in social networks, and their ability to utilize those positions for some goal. Designers need to enable people to sort themselves into comfortable positions in the social network, including leadership and follower positions.

► *Collective Intelligence* (or *Social Intelligence*) refers to the emergence of intelligent behavior among groups of

people. Designers can create mechanisms, such as voting systems, folksonomies, and other opinion aggregators, to ensure the emergence of social intelligence over time. (Note that the definition for “social intelligence” here differs from traditional use of the phrase in social psychology.)

The principal concern for designers of systems is to *ensure the participants both give and get something from the system that is beneficial to both the individual as well as to the group*. This may take the form of being challenged in their ideas, or to contribute to the overall knowledge of a domain, or to contribute their experiences of using a particular product or drug.

More importantly, social participation systems should encourage users to take part in effective action. One main design principle here is that *effective action arises from collective action*. That is, by encouraging participants to learn from each other and to form consensus, group goals will form, and action can be taken by the entire group.

The DARPA Network Challenge is interesting in that it was designed to see how we can get groups of people to take part in effective action. In that sense, the experiment was really quite successful. But we already have quite a good example of this in Wikipedia, in which a group of people came together to learn from each other’s perspective, but they share a common goal to create an encyclopedia of the state of human knowledge for broader distribution. Here, collective action resulted in effective change in the way people access information.

Looking toward the next decade, the social computing research challenge is understanding how to replicate effective social actions in social participation systems, in domains such as health care, education, and open government. United, we might just solve some of the biggest problems in the world.



### Mark Guzdial’s “Are There Too Many IT Jobs or Too Many IT Workers?”

<http://cacm.acm.org/blogs/blog-cacm/67389>

The latest U.S. Bureau of Labor Statistics (BLS) have been updated, as of November 2009, to reflect the Great

Recession. The news is terrific for us—computing is only forecast to grow, and at an amazing rate.

Via the Computing Community Consortium blog: “‘Computer and mathematical’ occupations are projected to grow by the largest percentage between now and 2018—by 22.2%. In other words, ‘Computer and mathematical’ occupations are the fastest-growing occupational cluster within the fastest-growing major occupational group.”

DARPA is so concerned about the lack of IT workers (and the lack of diversity among its workers) that it has launched a new research project to develop more and more diverse IT workers.

DARPA has launched a “far-out research” project to increase the number of students going into “CS-STEM” (computer science and science, technology, engineering, and mathematics) fields. *Wired* just covered this effort to address the “Geek shortage.” What makes the *Wired* piece so interesting is the enormous and harsh pushback in the comments section, like the below:

*“I’m 43, with a degree in software engineering and enjoy what I do for a living. But I wouldn’t encourage my 12-year-old son to major in CS or similar because interesting, new project development jobs are the first to disappear in a down economy and non-cutting-edge skills are easily offshored and new hires are cheaper than retraining outdated workers.”*

*“Why get a four-year degree for a career with a 15-year shelf life?”*

Are these complaints from a vocal small group, or are do they represent a large constituency? Why is there this disconnect between claims of great need and claims of no jobs? Are old IT workers no longer what industry wants? Is BLS only counting newly created jobs and not steady-state jobs? Is the IT job market constantly churning? Is industry not training existing people and instead hiring new people? It’s a real problem to argue for the need for more IT in the face of many (vocal) unemployed IT workers. □

Greg Linden is the founder of Geeky Ventures. Ed H. Chi is a research manager at Palo Alto Research Center. Mark Guzdial is a professor at the Georgia Institute of Technology.

© 2010 ACM 0001-0782/10/0600 \$10.00



DOI:10.1145/1743546.1743552

David Roman

## Interact Naturally

The mouse's days are numbered. Computer interfaces that remove user-system barriers are in the works and are intuitive enough for first-time users to throw away the manual. The iPhone's multitouch interface may have ushered in a wave of easier interfaces for the mass market, but it's just the beginning. Many new and exciting replacements for the familiar point-and-click scheme are on the way.

Skinput technology (<http://www.chrisharrison.net/projects/skinput/>) showcased at CHI 2010 (<http://cacm.acm.org/news/83935>) "appropriates" the human body as an input surface, says Carnegie Mellon Ph.D. student Chris Harrison, who developed SkinPut with Desney Tan and Dan Morris of Microsoft Research.

Gaze-based interfaces are being considered for data input, search, and selection (<http://portal.acm.org/toc.cfm?id=1743666&type=proceeding&coll=GUIDE&dl=GUIDE&CFID=86057285&CFTOKEN=34856226>), and driving vehicles (<http://cacm.acm.org/news/88018-car-steered-with-drivers-eyes/fulltext>). Voice controls have boarded Ford cars (<http://www.fordvehicles.com/technology/sync/>) and Apple smartphones.

Gesture interfaces is another hot area in the interface arena. MIT Media Lab Ph.D. candidate Pravan Mistry's Sixth Sense (<http://www.pranavmistry.com/projects/sixthsense/>) gesture interface (<http://cacm.acm.org/news/23600>) received a standing ovation when it premiered at the TED conference in February



**Sixth Sense: Using palm for dialing a phone number.**

2009 ([http://www.ted.com/talks/pattie\\_maes\\_demos\\_the\\_sixth\\_sense.html](http://www.ted.com/talks/pattie_maes_demos_the_sixth_sense.html)). Using multitouch gestures like the iPhone, Sixth Sense does not require a dedicated screen, but like many advanced interfaces it does depend on specialized hardware. Microsoft's Project Natal gesture interface ([http://en.wikipedia.org/wiki/Project\\_Natal](http://en.wikipedia.org/wiki/Project_Natal)) will give gamers hands-free control of the Xbox 360 in time for

the holidays season. There are dozens of related YouTube videos at <http://www.youtube.com/user/xboxprojectnatal>. Its application outside gaming is not clear.

Another promising but challenging area is the brain-machine interface (<http://cacm.acm.org/news/73070-building-a-brain-machine-interface/fulltext>), which sounds less fact than fiction, but was in fact the focus of DARPA's Augmented Cognition program (<http://www.wired.com/dangerroom/2008/03/augcog-continue/>).

All these interfaces aim to give users a simple, natural way to interact with a system. Microsoft's Chief Research and Strategy Officer Craig Mundie says natural user interfaces will appear first with gaming and entertainment systems but "will certainly find application...in the communications domain."

To read more about the interfaces of the future, check out the newly revamped ACM student magazine *XRDS* (formerly *Crossroads*). The print edition is available now; look for magazine's new Web site coming soon.

## ACM Member News

### ED LAZOWSKA WINS ACM'S DISTINGUISHED SERVICE AWARD



Ed Lazowska, the Bill & Melinda Gates Chair in Computer Science & Engineering

and director of the eScience Institute at the University of Washington, is the 2009 recipient of ACM's Distinguished Service Award for his wide-ranging service to the computing community and his long-standing advocacy for this community at the national level.

"Forty years ago, in 1969, there were three landmark events: Woodstock, Neil Armstrong's journey to the surface of the moon, and the first packet transmission on ARPANET," Lazowska said in an email interview. "With four decades of hindsight, which had the greatest impact? Unless you're big into Tang and Velcro (or sex and drugs), the answer is clear. Our future is every bit as bright as our past. No field is more important to the future of our nation or our world than computer science. We need to get that message out. Advances in computing are fundamental to addressing every societal grand challenge.

"I was recently party to a discussion with Anita Jones and Robin Murphy on 'Type I' and 'Type II' research in CS. Type I focuses on the technologies—compilers, operating systems, sensors. Type II focuses on the problems—engineering the new tools of scientific discovery, global development, health IT, emergency informatics, entertainment technology. I'm a Type II person. Either we expand our definition of CS to embrace these challenges, or we become insignificant.

"Forty years ago, as an undergraduate at Brown University, I was seduced into computer science by Andy van Dam. The potential to change the world is greater today than it has ever been. We need to communicate this to students. This field makes you powerful."

—Jack Rosenberger

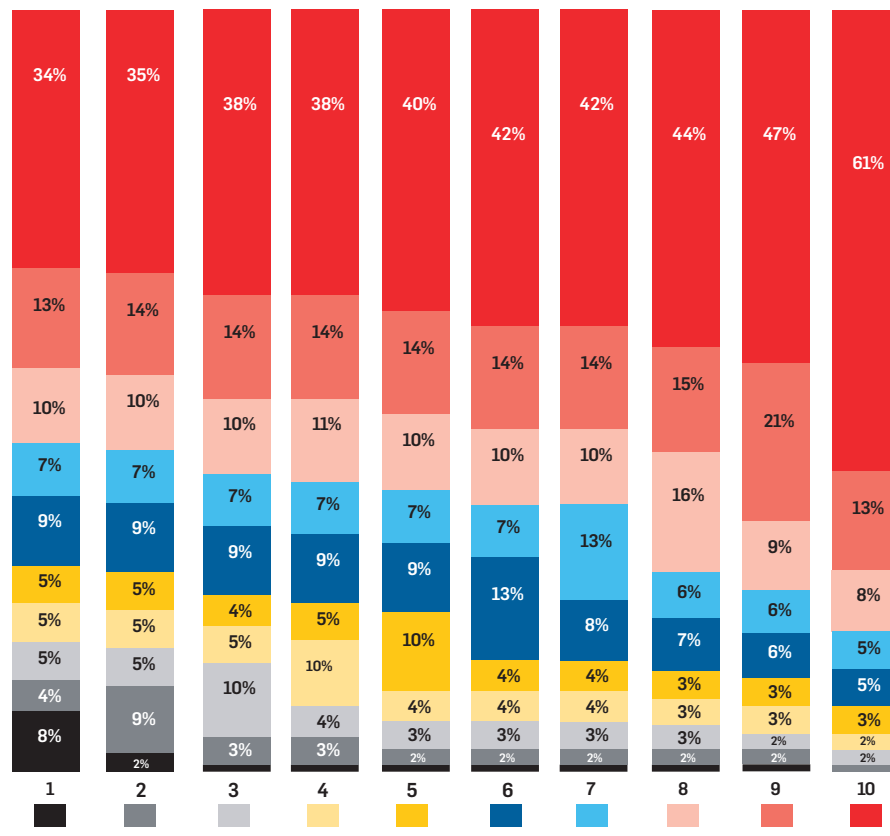
## Straightening Out Heavy Tails

*A better understanding of heavy-tailed probability distributions can improve activities from Internet commerce to the design of server farms.*

**O**NLINE COMMERCE HAS affected traditional retailers, moving transactions such as book sales and movie rentals from shopping malls to cyberspace. But has it fundamentally changed consumer behavior?

*Wired* Editor-in-Chief Chris Anderson thinks so. In his 2006 book titled *The Long Tail* and more recently on his Long Tail blog, Anderson argues that online retailers carry a much wider variety of books, movies, and music than traditional stores, offering customers many more products to choose from. These customers, in turn, pick more of the niche products than the popular hits. While individual niche items may not sell much more, cumulatively they're a bigger percentage of overall business.

The book's title comes from the shape of a probability distribution graph. Many phenomena follow the normal or Gaussian distribution; most of the values cluster tightly around the median, while in the tails they drop off exponentially to very small numbers. Events far from the median are exceedingly rare, but other phenomena, including book and movie purchases, follow a different course. Values in this distribution drop much less rapidly,



Each vertical bar represents a decile of DVD popularity, with the DVDs in decile 10 being the most popular. Each bar is subdivided to demonstrate how, on average, customers who rented at least one DVD from within that decile distributed their rentals among all the deciles. Shoppers in the bottom decile, for instance, selected only 8% of their rentals from among its titles—and 34% from among top-decile titles.

REPRINTED BY PERMISSION OF HARVARD BUSINESS REVIEW FROM "SHOULD YOU INVEST IN THE LONG TAIL?" BY ANITA ELBERSE, JULY-AUGUST 2008. COPYRIGHT 2008 BY THE HARVARD BUSINESS SCHOOL PUBLISHING CORPORATION. ALL RIGHTS RESERVED.

and events of roughly equal probability stretch far out into the tail.

Plotted on a graph, such distributions produce a peak very near the y-axis, a cluster near the intersection of the axes, and a long tail along the x-axis. This gives rise to the name long tail, or heavy tail. While a long tail is technically just one class of a heavy tail distribution—other terms include fat tail, power law, and Pareto distribution—in popular usage and for practical purposes there's no notable difference.

Because many phenomena have heavy tails—computer job sizes, for instance, and Web page links—researchers want to know how the distributions work and their effects.

Recently, for instance, business experts have tested the Long Tail theory and found that its effect on Internet commerce may not be as straightforward as some thought. Serguei Netessine, associate professor of operations and information management at the University of Pennsylvania, looked at data from Netflix, the movie rental service, containing 100 million online ratings of 17,770 movies, from 2000 to 2005, by 480,000 users. Netessine used the ratings as a proxy for rentals in his study, but says he has since looked at actual rental data.

Netessine's conclusion is, when looked at in percentage terms, the demand for hits actually grew, while the demand for niche products dropped. "There is no increased demand for bottom products that we are seeing in this data," Netessine says.

**"There might be an increasingly long tail, but that tail is getting thinner," says Anita Elberse.**

Anita Elberse, associate professor of business administration at Harvard Business School, earlier looked at data from Quickflix, an Australian movie rental service similar to Netflix, and from Nielsen VideoScan and Nielsen SoundScan, which monitor video and music sales, respectively, and reached the same conclusion. "There might be an increasingly long tail, but that tail is getting thinner," Elberse says.

Anderson disputes their conclusions, saying the researchers define the Long Tail differently. Anderson defines hits in absolute terms, the top 10 or top 100 titles, while the academics look at the top 1% or 10% of products. "I never do percentage analysis, since I think it's meaningless," Anderson wrote in an email interview. "You can't say 'I choose to define Long Tail as  $X$  and  $X$  is wrong, therefore Chris Anderson is wrong.' If you're going to critique the theory, you've got to actually get the theory right."

Anderson points to the same Netflix data used by Netessine and notes that

the top 500 titles dropped from more than 70% of demand in 2000 to under 50% in 2005. In 2005, Anderson notes, 15% of demand came from below the top 3,000, about the point where brick-and-mortar stores run out of inventory.

"In that sense, Anderson was right, but to me that was more or less a trivial finding, because each year we have more and more movies available," Netessine says.

### Influencing Consumers' Choices

An improved understanding of where in the distribution consumers land could lead to new methods of swaying their choices, in the form of better-designed recommendation engines. Researchers postulate that one reason consumers fail to choose niche products is that they have no way to find them. "If nobody is buying these items, how do they get recommended in the first place?" asks Kartik Hosanagar, associate professor of operations and information management at the University of Pennsylvania.

Hosanagar says collaborative filtering, based on user recommendations, can't find undiscovered items. It can, however, find items that are somewhat popular, and bring them to the attention of customers who might have missed them. He found that consumers appreciated recommendation engines that suggest more niche items, perhaps because they were already aware of the blockbusters.

He says retailers might boost their sales with improved recommendation engines. Using content analysis, which

## Obituary

# PC Pioneer Ed Roberts, 1941–2010

Henry Edward "Ed" Roberts, who created the first inexpensive personal computer in the mid-1970s, died on April 1 at the age of 68. Roberts is often credited as being "the father of the personal computer."

Roberts and a colleague founded Micro Telemetry Instrumentation Systems (MITS) in 1970 to sell electronics kits to model-rocket hobbyists. In the mid-1970s, MITS developed the Altair 8800, a programmable computer, which sold for a starting price of \$397. The Altair

8800 was featured on the January 1975 cover of *Popular Electronics*, and MITS shipped an impressive 5,000 units within a year.

The *Popular Electronics* cover story caught the attention of Paul Allen, a Honeywell employee, and Bill Gates, a student at Harvard University. They approached Roberts and were soon working at MITS, located in Albuquerque, NM, where they created the Basic programming language for the Altair 8800. It was a move that would ultimately lead to the founding of Microsoft Corp.

MITS was sold to Pertec Computer Corporation in 1977, and Roberts received \$2 million. He retired to Georgia and first worked as a farmer, then studied medicine and became a physician.

Agreement about who invented the first personal computer differs, with credit being variously given to Roberts, John Blankenbaker of Kenbak Corporation, Xerox Palo Alto Research Center, Apple, and IBM. Roberts' impact on computing, though short in

duration, is immeasurable. "He was a seed of this thought that computers would be affordable," Apple cofounder Steve Wozniak has said. The breakthrough insight for Roberts might have occurred during the late 1960s while working on a room-size IBM computer as an electrical engineering major at Oklahoma State University. As he later said in an interview, "I began thinking, What if you gave everyone a computer?"

—Jack Rosenberger



identifies characteristics of a product—say, the director or genre of a movie—and suggesting it to buyers of products with similar characteristics, could increase the diversity of recommendations. Hosanagar says researchers looking at Internet commerce shouldn't assume sales mechanisms and buyers' behavior are unchangeable. "A lot of social scientists are looking at the system as a given and trying to look at its impact, but what we are saying is the system is not a given and there are a lot of design factors involved," he says.

That matches the thinking of Michael Mitzenmacher, a professor of computer science at Harvard, who says researchers need a better understanding of how heavy tails come to be, so that they can turn that knowledge to practical use. "For a long time people didn't realize power law distributions come up in a variety of ways, so there are a variety of explanations," Mitzenmacher says. "If we understand the model of how that power law comes to be, maybe we can figure out how to push people's behavior, or computers' behavior, in such a way as to improve the system."

For instance, file size follows a power law distribution. An improved understanding of that could lead to more efficiently designed and economical file storage systems. Or if hyperlinks are similar to movie rentals—that is, if the most popular Web pages retain their popularity while the pages out in the tail remain obscure—it might make sense to take that into account when designing search engines. And if search engines have already changed that dynamic, it could be valuable to understand how.

One area where heavy tails affect computer systems is the demand that UNIX jobs place on central processing units (CPUs). Mor Harchol-Balter, associate department head of graduate education in the computer science department at Carnegie Mellon University, says the biggest 1% of jobs make up half the total load on CPUs. While most UNIX jobs may require a second or less of processing time, some will need several hours.

If there were low variability among job sizes, as people used to believe, it would make sense to have just a few, very fast servers. But because the job size distribution is heavy tailed, it's more ef-

ficient to use more, slower machines. Designing server farms with this understanding, Harchol-Balter says, could cut electricity demand by 50%.

"In computer science, we really concentrate on understanding the distribution of the sizes of the requirements, and making our rules based on this understanding," she notes. "We are having to invent whole new mathematical fields to deal with these kinds of distributions."

Harchol-Balter finds the origin of heavy tail distributions a fascinating question, one she sometimes asks her classes to speculate about. "Why are files at Web sites distributed according to a Pareto distribution? Why on Earth? Nobody set out to make this kind of distribution," she says. "Somehow most people write short programs and some people write long programs and it fits this distribution very well."

But the "why" isn't her main concern. "I don't need to know why it happens," Harchol-Balter says. "I just need to know if it's there what I'm going to do with it." □

#### Further Reading

*Tan, T.F. and Netessine, S.*

Is Tom Cruise threatened? Using Netflix Prize data to examine the long tail of electronic commerce. Working paper, University of Pennsylvania, Wharton Business School, July 2009.

*Elberse, A.*

Should you invest in the long tail? *Harvard Business Review*, July-August 2008.

*Mitzenmacher, M.*

Editorial: the future of power law research. *Internet Mathematics* 2, 4, 2005.

*Harchol-Balter, M.*

The effect of heavy-tailed job size distributions on computer system design. *Proceedings of ASA-IMS Conference on Applications of Heavy Tailed Distributions in Economics, Engineering and Statistics*, Washington, DC, June 1999.

*Fleder, D.M. and Hosanagar, K.*

Blockbuster culture's next rise or fall: the impact of recommender systems on sales diversity. *Management Science* 55, 5, May 2009.

*Newman, M.E.J.*

Power laws, Pareto distributions, and Zipf's law. *Contemporary Physics* 46, 323–351, 2005.

**Neil Savage** is a science and technology writer based in Lowell, MA. Prabhakar Raghavan, Yahoo! Research, contributed to the development of this article.

© 2010 ACM 0001-0782/10/0600 \$10.00

## Networking

# Quantum Milestone

Researchers at Toshiba Research Europe, in Cambridge, U.K., have attained a major breakthrough in quantum encryption, with their recent continuous operation of quantum key distribution (QKD) with a secure bit rate of more than 1 megabit per second over 50km of fiber optic cable. The researchers' feat, averaged over a 24-hour period, is 100–1,000 times higher than any previous QKD for a 50km link. The breakthrough could enable the widespread usage of one-time pad encryption, a method that is theoretically 100% secure.

First reported in *Applied Physics Letters*, the QKD milestone was achieved with a pair of innovations: a unique light detector for high bit rates and a feedback system that maintains a high bit rate and, unlike previous systems, does not depend on manual set-up or adjustments.

"Although the feasibility of QKD with megabits per second has been shown in the lab, these experiments lasted only minutes or even seconds at a time and required manual adjustments," says Andrew Shields, assistant managing director at the Cambridge lab. "To the best of our knowledge this is the first time that continuous operation has been demonstrated at high bit rates. Although much development work remains, this advance could allow unconditionally secure communication with significant bandwidths."

The QKD breakthrough will allow the real-time encryption of video with a one-time pad. Previously, researchers could encrypt continuous voice data, but not video.

Toshiba plans to install a QKD demonstrator at the National Institute of Information and Communications Technology in Tokyo. "The next challenge would be to put this level of technology into metropolitan network operation," says Masahide Sasaki, coordinator of the Tokyo QKD network. "Our Japan-EU collaboration is going to do this within the next few years."

# Beyond the Smart Grid

*Sensor networks monitor residential and institutional devices, motivating energy conservation.*

**A**S PRESIDENT-ELECT, BARACK Obama used the term “smart grid” in his first major speech of 2009, and few phrases have enjoyed as much currency recently. The electrical grid isn’t the only utility acquiring intelligence, however, as water and gas meters throughout the U.S. gain radio communication capabilities and other innovations.

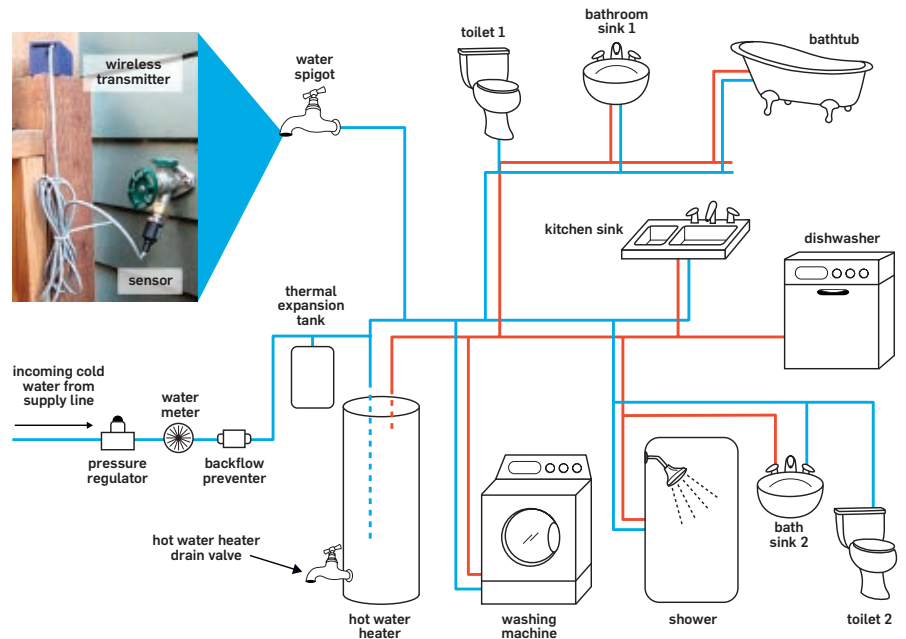
But those grids (and their attendant smartness) stop at the residential meter, so consumers never know which household devices are the biggest energy users. Once monitored, these devices would need to communicate—to turn on the ceiling fan and adjust the air conditioner when electricity prices peak, for example. The final ingredient for such a system to be useful to consumers is an easy-to-understand interface for monitoring and controlling the devices.

The solution, like the problem, has three parts. First, monitor each device separately; second, network them together for coordination; third, present the resulting data in an easy-to-use format. As it happens, this solution goes beyond energy conservation to suggest new ways of integrating home automation, safety, security, and entertainment applications with smart grid data.

## Your Home’s Five Senses

The first key part is the sensors themselves. For utility monitoring, installation has been a major barrier to consumer adoption. Measuring water flow to a specific faucet, for example, required removing a section of pipe. To give a residential consumer the whole picture, this process would have to be repeated for every faucet in the house.

But now work done by Shwetak Patel, an assistant professor in the department of computer science and engineering at University of Washington,



**HydroSense can be installed at any accessible location in a home’s water infrastructure, with typical installations at an exterior hose bib, a utility sink spigot, or a water heater drain valve. By continuously sensing water pressure at a single installation point, HydroSense can identify individual fixtures where water is being used and estimate their water usage.**

and colleagues can extrapolate electrical, water, and gas use of individual devices by measuring the “shock waves” created when consumers turn on the devices that use those utilities.

Patel’s HydroSense approach is to attach a single sensor to a spigot and examine differences in pressure created by the water-hammer phenomenon when individual devices are turned on and off. After building a profile of devices in the house, he says, the single sensor can accurately tell each device from the others within a 5% margin of error. The same model works for gas lines as well; for electricity, the single plug-in sensor looks for characteristic noise patterns produced by individual devices over the home’s electrical lines.

Patel points out the educational value of this information. “Often peoples’ mental model of [utility] consumption is really inaccurate,” he says. “Over 30 years of study in environmental psy-

chology has shown that giving people itemized feedback can reduce overall energy use by 15%–20%. But adoption of sensors that will give them that feedback drastically drops off as the installation burden increases. So the question is, How can we build a single sensor that gives them disaggregated information, but doesn’t need a professional electrician or plumber to install? If we can build cheap sensors that give consumers effective feedback, they can start to reduce overall consumption in their home.”

Even with single-point sensors installed, there’s still a place for individual sensors to measure environmental factors. For example, a sensor that measures the oil level in a furnace could switch on electric heating when the oil is running out, but only during times of low electricity demand. Or a combustible-gas sensor could prevent an explosion, when a gas leak is detected, by preventing a gas furnace’s igni-

tion from sparking on. Such concepts require a development platform that hosts both sensors and communications hardware.

One such platform is SquidBee and its successor, Wasmote. Both originated at the Spain-based company Libelium, which also produces three types of Wasmote sensor boards. One determines the presence of gases, such as carbon dioxide and methane; the second senses environmental changes, such as vibration, pressure, and moisture; and the third is a prototype board that will host any other sensor types a developer might have. For applications that don't require immediate communication or in situations where immediate communication is impossible, the Wasmote board contains two gigabytes of internal memory for later transmission.

### Making Sensors Talk

Both Patel and Libelium's devices require a way to communicate their findings to the outside world. Wasmote uses a variety of methods, including USB, GPS, 802.15.4, and a range of radio frequencies. Patel is agnostic about the communication methods his still-in-development devices will use. "We're innovating on the hardware, aggregation, and signal processing," he says, "but not on the network."

One specification that both plan to use is ZigBee, an extension of the 802.15.4 standard promoted by the nonprofit, industry-based ZigBee Alliance. According to ZigBee Alliance Chairman Bob Heile, ZigBee was designed specifically "to create open, global standards for wireless sensor networks." As such, it prioritizes power consumption and transmission integrity so that the devices—which might be used in difficult-to-access areas—can operate trouble-free for a long period of time. "We're achieving devices that go for five to 10 years on an alkaline battery or 10 to 20 years on lithium-ion," says Heile.

The ZigBee Alliance also prioritized scalability well beyond the residential needs. Heile says the ARIA Resort & Casino in the new CityCenter development in Las Vegas has more than 90,000 ZigBee-compliant devices to control both common-area and guest-room environments. On the other

## Oberlin College hosts an annual dorm energy competition, with prizes for the dorm that achieves the greatest energy reduction.

hand, ZigBee largely ignores issues of bandwidth and quality of service, as would be needed for a telephony or video application.

The ZigBee specifications cover all seven layers of the Open Systems Interconnection model, in three parts. The bottom two—the physical and data-link layers—are the 802.15.4 standard, with no changes. Layers three to six comprise the "ZigBee stack," including algorithms for organization among nodes, error routing, and AES-128 security. (As a wireless technology, the security portion is especially important to prevent outside tampering that could cause unpredictable device behavior.) When layers one through six are implemented according to the ZigBee specification, it qualifies for ZigBee platform compliance certification. ZigBee-certified products also implement layer seven, which is a ZigBee public profile such as smart energy, home automation, or health care.

### Acting on Data

Once the data is collected, it needs to be presented in ways that are understandable to humans and to other devices. "We don't want to overwhelm the consumer with a bunch of data," says Patel. "We could provide them with a 'Top Ten Energy Consumers in Your Home' list to give them something to work on. Or if we see that the compressor in their refrigerator is degrading in performance over time, we could give them targeted advice on blowing out the coils."

One example of how such data is being used is found in Oberlin College's campus resource monitoring system. The environmental studies program

monitors electricity use in each of the college's dorms, in some cases with multiple sensor points per dorm. Administrators make adjustments to discount nondiscretionary expenditures, such as a kitchen in those dorms with cafeterias, then take a baseline reading to determine typical usage. Data from dorms' current energy use is displayed in three ways: on the Web at oberlin.edu/dormenergy; as building dashboard video displays throughout campus; and as color-changing orbs placed in several campus locations, including the dorms themselves.

Finally, Oberlin College runs an annual dorm energy competition and gives prizes to the dorm with the greatest reduction from baseline use. Henry Bent, sustainable technology research fellow partly responsible for maintaining the Oberlin system, is especially enthusiastic about the orbs. "Numbers and dials and graphs are fantastic, but you want something that you can see very quickly at a glance," Bent says. "I just know when I'm on my way to the bathroom, 'Oh, look, that orb is red, I should turn something off.' "

### Further Reading

Patel, S.N., Robertson, T., Kientz, J.A., Reynolds, M.S., Abowd, G.D.

At the flick of a switch: detecting and classifying unique electrical events on the residential power line. *Proceedings of Ubicomp 2007*, Innsbruck, Austria, September 16–19, 2007.

Patel, S.N., Reynolds, M.S., Abowd, G.D.

Detecting human movement by differential air pressure sensing in HVAC system ductwork: an exploration in infrastructure mediated sensing. *Proceedings of Pervasive 2008*, Sydney, Australia, May 19–22, 2008.

Petersen, J.E., Shunturov, V., Janda, K., Platt, G., Weinberger, K.

Dormitory residents reduce electricity consumption when exposed to real-time visual feedback and incentives. *International Journal of Sustainability in Higher Education* 8, 1, 2007.

Fischer, C.

Feedback on household electricity consumption: a tool for saving energy? *Energy Efficiency* 1, 1, Feb. 2008.

Wireless Sensor Networks Research Group  
<http://www.sensor-networks.org>.

Tom Geller is an Oberlin, Ohio-based science, technology, and business writer.

© 2010 ACM 0001-0782/10/0600 \$10.00



# Mine Your Business

*Researchers are developing new techniques to gauge employee productivity from information flow.*

**W**HAT'S THE BEST way to measure employee productivity in the digital age? From lines of code to number of sales, each industry has its own imperfect standards. Yet according to a new line of thought, the answer may, in fact, lie in your email inbox. And your chat log. And the comments you added to a shared document—in the sum, that is, of your electronic activity. Researchers at IBM and Massachusetts Institute of Technology, for example, analyzed the electronic data of 2,600 business consultants and compared their communication patterns with their billable hours. The conclusion: the average email contact is worth \$948 in annual revenue. Of course, it also makes a difference who your contacts are. Consultants with strong ties to executives and project managers generated an average of \$7,056 in additional annual revenue, compared with the norm.

According to Redwood City, CA-based Cataphora, one of the companies at the forefront of the movement, the objective is to build patterns of activity—then flag and analyze exceptions.

“We’re interested in modeling behavior,” says Keith Schon, a senior software engineer at Cataphora. “What do people do normally? How do they deviate when they’re not acting normally?” Using data mining techniques that encompass network analysis, sentiment analysis, and clustering, Schon and colleagues analyze the flow of electronic data across an organization. “We’re trying to figure out relationships,” he explains.

Cataphora got its start in the electronic discovery field, where understanding what people know and how that knowledge spreads is critical to legal liability. The company thus works to uncover so-called “shadow networks” of employees who know each other through non-business channels like colleges or churches, or who share



**IBM's SmallBlue technology analyzes employees' electronic data and creates a networked map of who they're connected to and what their expertise is.**

a native language, and could collude with one another. Its engineers search for unusual linguistic patterns, and set actual communication networks against official organization charts to determine when people interact with those to whom they have no ostensible work connection.

Yet Cataphora and others are also developing tools to analyze such patterns of behavior in non-investigative settings in the hope of understanding—and enhancing—employee productivity. Microsoft examines internal communications to identify so-called “super connectors,” who communicate frequently with other employees and share information and ideas. Eventually, researchers say, that data could help business leaders make strategic decisions about a project team’s composition, effectiveness, and future growth. Likewise, Google is testing an algorithm that uses employee review data, promotions, and pay histories to identify its workers who feel underused, and therefore are most likely to leave the company. Though Google is reluctant to share details, human resources director Laszlo Bock has said the idea is to get inside people’s heads before they even

think about leaving—and to work harder to keep them engaged.

“We have access to unprecedented amounts of data about human activity,” says Sinan Aral, a professor of management sciences at New York University’s Stern School of Business who studies information flow. Of course, not every benefit an individual brings to a company can be captured electronically, “but we can explain a lot,” Aral says. Researchers hasten to add they’re not seeking to punish people for using Facebook at work or making personal phone calls. “The social stuff may be important, and we don’t count that against a person,” says Cataphora’s Schon. In most cases, in fact, personal communications are filtered out and ignored.

## Measuring Electronic Productivity

Some measures of electronic productivity are relatively straightforward. Cataphora, for example, seeks to identify blocks of text that are reused, such as a technical explanation or a document template, reasoning that the employees who produce them are making a comparatively greater impact on the company by doing work that others deem valuable. Such text blocks can be difficult to identify, for their language often



evolves as they spread through a corporate network. Cataphora has developed a fuzzy search algorithm to detect them, but Schon admits the task is complex. Creating an algorithm that organizes sentences into text blocks, for example, often forces researchers to make inflexible choices about boundaries, using punctuation, length limits, paragraph breaks, or some other scheme. That, in turn, could cause a program to overlook a document whose author formats things differently, such as not breaking the text into paragraphs very frequently or using unconventional punctuation.

Cataphora has also developed a proprietary set of ontologies that cover human resources-related topics, marketing issues, product development, and more to examine various subject-specific communications. One way in which they are useful, Schon explains, is for studying the relationships between people and topics. If an executive is central to communications about product development, marketing, and finance, but marginal to those about sales, it's likely that she or he is out of the loop when it comes to the newest sales tactics. Ontologies can also identify communications related to particular tasks, such as hiring and performance reviews. From there, engineers can statistically determine what the "normal" procedure is, and see when it is and isn't followed. Thanks to the training corpus Cataphora has built over time through its clients, these ontologies perform quite well. Yet to detect communication that is specific to a particular industry, location, or research group and whose names can be idiosyncratic, "we may need to examine the workflow and develop more specific ontologies," says Schon.

Further analysis helps identify how employees influence each other at work. Aral, for example, correlates his electronically derived network topologies with traditional accounting and project data, such as revenues and completion rates, to try to understand which factors enhance or diminish certain outcomes. "The old paradigm was that each employee had a set of characteristics, like skills or education, which he or she brought to a firm," Aral explains. "Our perspective is that employees are all connected, and that companies build a network of individ-

## Microsoft examines internal communications to identify so-called "super connectors," who communicate frequently with fellow employees and share information and ideas.

uals who help each other." In a study of five years of data from an executive recruiting firm, Aral found that employees who were more central to the firm's information flow—who communicated more frequently and with a broader number of people—tended to be more productive. It makes a certain amount of sense. "They received more novel information and could make matches and placements more quickly," Aral notes. In fact, the value of novel information turned out to be quite high. Workers who encountered just 10 novel words more than the average worker were associated with an additional \$70 in monthly revenue.

Yet Aral's conclusions also point to one of the more challenging aspects of this type of research. If a position in the corporate network is associated with increased productivity, is it because of the nature of that position or because certain kinds of people naturally gravitate toward it? "You always have to question your assumptions," admits Aral. New statistical techniques are needed, he says, to more accurately distinguish correlation from causation.

Large-scale data mining presents another challenge. IBM's SmallBlue, which grew out of research at its Watson Business Center, analyzes employees' electronic data and creates a networked map of who they're connected to and where their expertise lies. Employees can then search for people with expertise on certain subjects and find the shortest "social path" it would

take to connect them. SmallBlue is an invaluable tool for large, international firms, and IBM has used it to connect its 410,000 employees since 2007. Yet indexing the 20-plus million emails and instant messages those employees write is not a trivial task—not to mention the 2 million blog and database entries and 10 million pieces of data that come from knowledge sharing and learning activities. It is the largest publicly known social network dataset in existence, and the project's founder, Ching-Yung Lin, says IBM worked hard to design a database that would hold different types of data and dynamically index the graphs that are generated.

Proponents of electronic productivity analysis say the markers are best used to augment, rather than replace, traditional metrics and peer evaluations. "It's a sanity check," asserts Schon. In the future, predicts Aral, who is helping IBM refine SmallBlue, the software could provide real-time, expertise-based recommendations: automatically suggesting connections while employees work on a particular task, for example, or helping managers assemble compatible project teams. ■

### Further Reading

Aral, S., Brynjolfsson, E., and Van Alstyne, M. Information, technology and information worker productivity. *International Conference on Information Systems*, Milwaukee, WI, 2006.

Manning, C.D., Raghavan, P., and Schütze, H. *Introduction to Information Retrieval*. Cambridge University Press, New York, 2008.

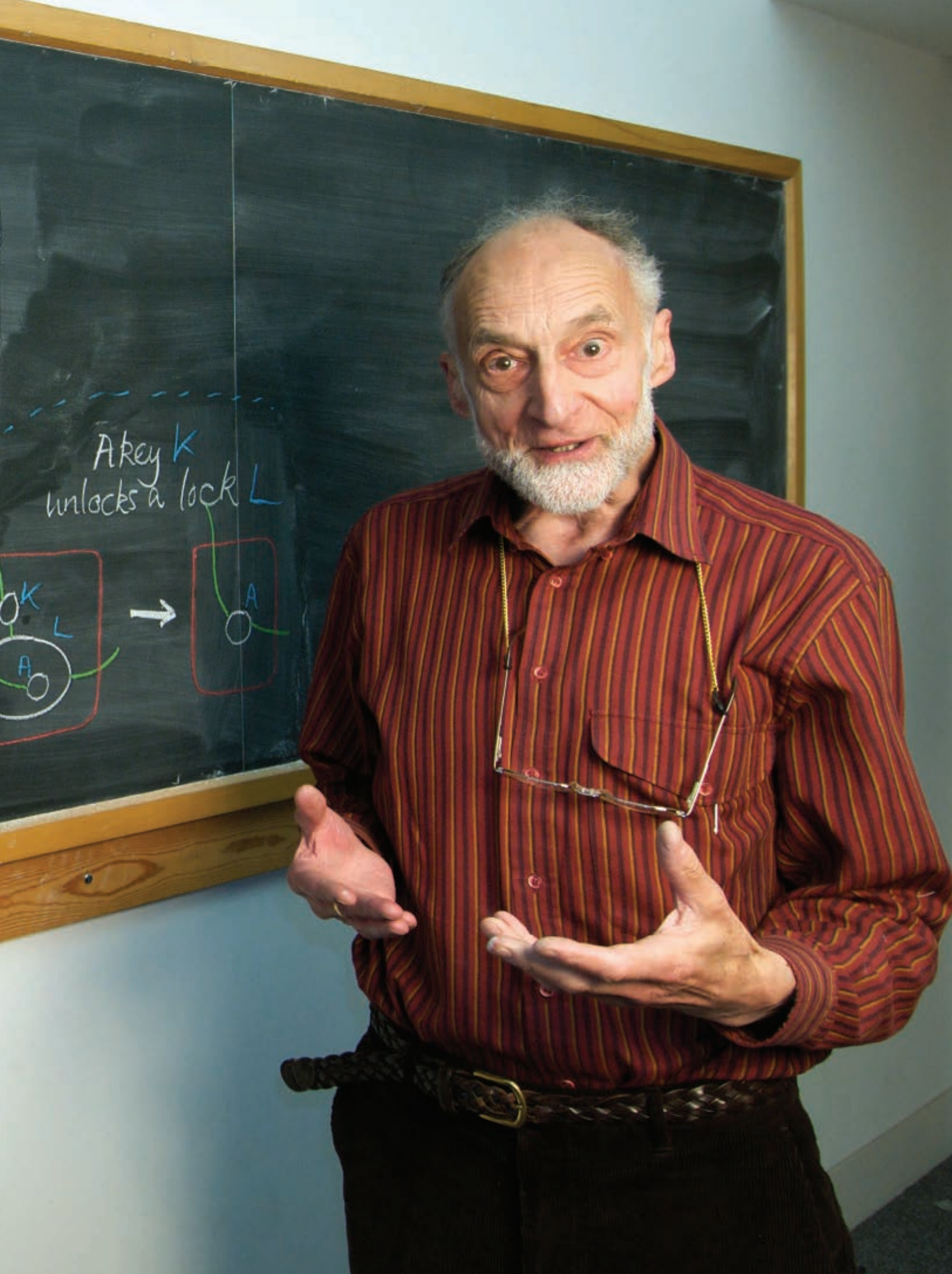
Mikawa, S., Cunningham, S., and Gaskis, S. Removing barriers to trust in distributed teams: understanding cultural differences and strengthening social ties. *International Workshop on Intercultural Collaboration*, Palo Alto, CA, 2009.

Wasserman, S. and Faust, K. *Social Network Analysis: Methods and Applications*. Cambridge University Press, Cambridge, 1994.

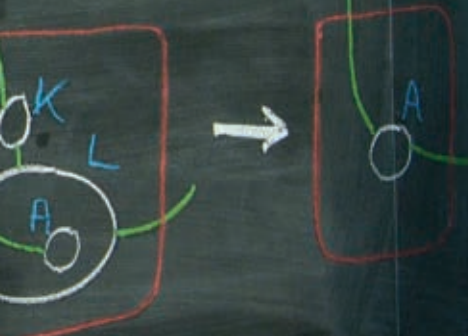
Wu, L., Lin, C.-Y., Aral, S., and Brynjolfsson, E. Value of social network: a large-scale analysis on network structure impact to financial revenue of information technology consultants. *Winter Information Systems Conference*, Salt Lake City, UT, 2009.

Leah Hoffmann is a Brooklyn, NY-based technology writer.

© 2010 ACM 0001-0782/10/0600 \$10.00



A key  $K$   
unlocks a lock  $L$





# Robin Milner: The Elegant Pragmatist

*Remembering a rich legacy in verification, languages, and concurrency.*

**I**T CAME AS a surprise even to those who knew him well: the death of Arthur John Robin Gorell Milner, known to friends simply as Robin, of a heart attack on March 20. Milner's wife, Lucy, had died three weeks before. Yet the pioneering computer scientist remained active, and seemingly in good health, until the end, maintaining close connections with his colleagues and even co-authoring a paper with a postdoctoral student he supervised at the IT University of Copenhagen.

A man of modest background and quiet brilliance, Milner made groundbreaking contributions to the fields of verification, programming languages, and concurrency. He was born in 1934 near Plymouth, England, and won scholarships to Eton—where he developed an enduring love of math as well as a prodigious work ethic—and King's College, Cambridge. It was during his time at Cambridge that Milner was introduced to programming, though the subject didn't interest him initially. "I regarded programming as really rather inelegant," he recalled in an interview in 2001 with Martin Berger, a professor at the University of Sussex. "So I resolved that I would never go near a computer in my life." Several years later, in 1960, Milner broke that resolution with a programming job at Ferranti, a British computer company; from there, it wasn't long before he moved to academia with positions at City University London, Swansea University, Stanford University, and the Universities of Edinburgh and Cambridge.

Inspired by Dana Scott's famous formulation of domain theory, Milner began working on an automatic theorem prover, hoping to find a way to mechanize a logic for reasoning about programs. The work culminated in the development of Logic for Computable

**"Robin initiated a revolution in what computing languages are and could be," says Robert Harper.**

Functions (LCF), an interactive system that helps researchers formulate proofs. "It was a novel idea," says Robert Harper, a professor of computer science at Carnegie Mellon University. "The approach before then was that computers would search for your proof. Robin recognized that the computer is a tool to help you find the proof." During that research, Milner also laid the foundations of ML, a metalanguage whose original intent was to enable researchers to deploy proof tactics on LCF. The innovative concepts it introduced, however—such as polymorphic type inference and type-safe exception handling—were soon recognized. Ultimately, ML evolved into a powerful general programming language (with Milner, Harper, and others working to specify and standardize it) and led to languages like F# and Caml.


"Robin initiated a revolution in what computing languages are and could be," asserts Harper.

Although he was closely involved in ML's development throughout the 1980s and 1990s, Milner also began working on the problem of concurrency, looking for a mathematical treatment that could rival theories of sequential computation. The Calculus of Communicating Systems (CCS) was the first solution he devised: a process calculus for a network that was

programmed to cooperate on a single task. CCS was succeeded by a more general theory of concurrency called pi calculus, which incorporated dynamic generation, and bigraphs, a theory of ubiquitous computing. For his work on LCF, ML, and CCS, Milner received the ACM A.M. Turing Award in 1991.

"Robin had this ability to think large and translate that all the way down to new theorems," says Mads Tofte, vice chancellor of the IT University of Copenhagen and a former graduate student. The sentiment is echoed by many of Milner's collaborators, who cite his unfailing ability to shift from a grand vision of computing to subtle mathematical particulars. Coupled with his rigorous attention to detail, the trait gave Milner's work a firm grounding in practical applications. "He was very concerned with making things work," says Gordon Plotkin, a former colleague at the University of Edinburgh.

Milner also labored on behalf of institutions and initiatives that shaped the future of the field. He helped establish the Laboratory for Foundations of Computer Science at the University of Edinburgh and served as the first chair of the University of Cambridge's Computer Laboratory. In 2002, Milner and Tony Hoare, a senior researcher at Microsoft Research in Cambridge (and a fellow Turing Award winner), began working on a grand challenge initiative to identify research topics that would drive science in the 21st century.

He was a natural leader, according to Tofte. "I'm not sure he was always aware of it," says Tofte, "but he was so good at getting the best out of people, which is exactly what a good leader does." 

Leah Hoffmann is a Brooklyn, NY-based technology writer.

© 2010 ACM 0001-0782/10/0600 \$10.00

# CS and Technology Leaders Honored

**A**WARDS WERE RECENTLY announced by ACM and the American Association for the Advancement of Science honoring leaders in the fields of computer science and technology.

## ACM Awards

**VMware Workstation 1.0**, which was developed by Stanford University professor **Mendel Rosenblum** and his colleagues **Edouard Bugnion**, **Scott Devine**, **Jeremy Sugerman**, and **Edward Wang**, was awarded the Software System Award for bringing virtualization technology to modern computing environments, spurring a shift to virtual-machine architectures, and allowing users to efficiently run multiple operating systems on their desktops.

**Michael I. Jordan**, a professor at the University of California, Berkeley, is the recipient of the ACM/AAAI Allen Newell Award for fundamental advances in statistical machine learning—a field that develops computational methods for inference and decision-making based on data.

**Tim Roughgarden**, an assistant professor at Stanford University, received the Grace Murray Hopper Award for introducing novel techniques that quantify lost efficiency with the uncoordinated behavior of network users who act in their own self-interest.

**Matthias Felleisen**, a Trustee Professor at Northeastern University, was awarded the Karl V. Karlstrom Outstanding Educator Award for his visionary and long-standing contributions to K-12 outreach programs.

**Gregory D. Abowd**, a professor at Georgia Institute of Technology, is the recipient of the Eugene L. Lawler Award for Humanitarian Contributions within Computer Science and Informatics for promoting a vision of health care and education that incorporates the use of advanced information technologies to address difficult challenges re-



**Michael I. Jordan**, University of California, Berkeley

lating to the diagnosis and treatment of behavioral disorders, as well as the assessment of behavioral change within complex social environments.

**Edward Lazowska**, the Bill & Melinda Gates Chair in computer science and engineering at the University of Washington, received the Distinguished Service Award for his wide-ranging service to the computing community and his long-standing advocacy for this community at the national level. (An interview with Lazowska appears in the ACM Member News column on p. 12.)

**Moshe Y. Vardi**, the Karen Ostrum George Professor in Computational En-

**VMware Workstation 1.0 is the recipient of ACM's Software System Award.**

gineering and Director of the Ken Kennedy Institute for Information Technology at Rice University, is the recipient of the Outstanding Contribution to ACM Award for his leadership in restructuring ACM's flagship publication, *Communications of the ACM*, into a more effective communications vehicle for the global computing discipline, and for organizing an influential, systematic analysis of offshoring, *Globalization and Offshoring of Software*, which helped reinforce the case that computing plays a fundamental role in defining success in a competitive global economy.

**Elaine Weyuker** and **Mathai Joseph** were named recipients of the ACM Presidential Award. Weyuker, an AT&T Fellow at Bell Labs, was honored for her efforts in reshaping and enhancing the growth of ACM-W to become a thriving network that cultivates and celebrates women seeking careers in computing. Joseph, an advisor to Tata Consultancy Services, was honored for his commitment to establishing an ACM presence in India. His efforts were instrumental in the formation of the ACM India Council, which was launched last January.

## American Academy Fellows

Eight computing scientists and technology leaders were among the 229 newly elected Fellows and Foreign Honorary Members to the American Association for the Advancement of Science. They are: **Randal E. Bryant**, Carnegie Mellon University; **Nancy A. Lynch**, Massachusetts Institute of Technology; **Ray Ozzie**, Microsoft; **Samuel J. Palmisano**, IBM; **Burton Jordan Smith**, Microsoft; **Michael Stonebraker**, Vertica Systems; **Madhu Sudan**, Massachusetts Institute of Technology; **Moshe Y. Vardi**, Rice University; and **Jeannette M. Wing**, Carnegie Mellon University/National Science Foundation. □

**Jack Rosenberger** is *Communications'* senior editor, news.

© 2010 ACM 0001-0782/10/0600 \$10.00





# ACM Europe Council

## Serving the European Computing Science Community

ACM announces the ACM Europe Council: 16 leading European computer scientists from academia and industry to spearhead expansion of ACM's high-quality technical activities, conferences and services in Europe.

- ◆ Holding high-quality ACM conferences in Europe
- ◆ Expanding ACM chapters
- ◆ Strong co-operation with other European scientific societies in computing

*"Our goal is to share ACM's vast array of valued resources and services on a global scale. We want to discover the work and welcome the talent from all corners of the computing arena so that we are better positioned to appreciate the key issues and challenges within Europe's academic, research, and professional computing communities, and respond accordingly,"* says **Professor Dame Wendy Hall, U. of Southampton (UK), ACM President.**



### ACM – World's Largest Educational and Scientific Computing Society

Since 50 years, ACM strengthens the computing profession's collective voice through strong leadership, promotion of the highest standards, and recognition of technical excellence. ACM supports the professional growth of its members by providing opportunities for life-long learning, career development, and professional networking.

**Fabrizio Gagliardi, ACM Europe Chair and Director of External Research Programs, Microsoft Research Europe** says, *"By strengthening ACM's ties in the region and raising awareness of its many benefits and resources with the public and European decision-makers, we can play an active role in the critical technical, educational, and social issues that surround the computing community."*



### Supporting the European Computing Community

ACM Europe aims to strengthen the European computing community at large, through its members, chapters, sponsored conferences and symposia. Together with other scientific societies, it helps make the public and decision makers aware of technical, educational, and social issues related to computing.

### A European Perspective Within ACM

The ACM Europe Council brings a unique European perspective inside ACM and helps increase visibility of ACM across Europe, through:

- ◆ Participation of Europeans throughout ACM
- ◆ Representation of European work in ACM Awards and Advanced Membership Grades

ACM is present in Europe with 15 000 members and 41 chapters. 23 ACM Turing Awards and other major ACM awards have gone to individuals in Europe. 215 Europeans received an Advanced Membership Grade since 2004.

The ACM Europe Council represents European computer scientists. Contact us at: [acmeurope@acm.org](mailto:acmeurope@acm.org) or <http://europe.acm.org/>

# Privacy and Security

## Myths and Fallacies of “Personally Identifiable Information”

*Developing effective privacy protection technologies is a critical challenge for security and privacy research as the amount and variety of data collected about individuals increase exponentially.*

**T**HE DIGITAL ECONOMY relies on the collection of personal data on an ever-increasing scale. Information about our searches, browsing history, social relationships, medical history, and so forth is collected and shared with advertisers, researchers, and government agencies. This raises a number of interesting privacy issues. In today’s data protection practices, both in the U.S. and internationally, “personally identifiable information” (PII)—or, as the U.S. Health Insurance Portability and Accountability Act (HIPAA) refers to it, “individually identifiable” information—has become the *lapis philosophorum* of privacy. Just as medieval alchemists were convinced a (mythical) philosopher’s stone can transmute lead into gold, today’s privacy practitioners believe that records containing sensitive individual data can be “de-identified” by removing or modifying PII.

### What is PII?

For a concept that is so pervasive in both legal and technological discourse

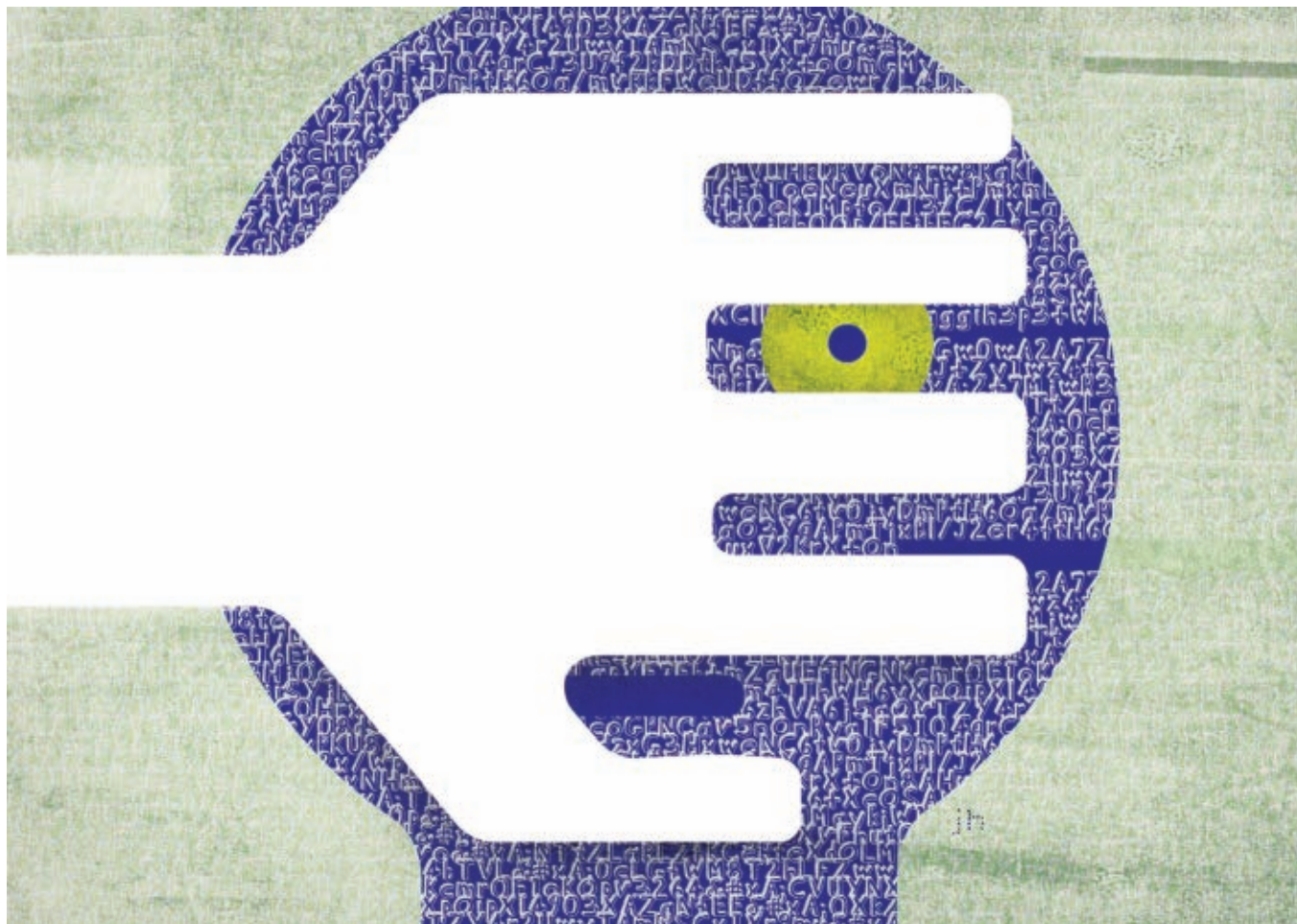
**Any information that distinguishes one person from another can be used for re-identifying data.**

on data privacy, PII is surprisingly difficult to define. One legal context is provided by breach-notification laws. California Senate Bill 1386 is a representative example: its definition of personal information includes Social Security numbers, driver’s license numbers, financial accounts, but not, for example, email addresses or telephone numbers. These laws were enacted in response to security breaches involving customer data that could enable identity theft. Therefore, they focus solely on the types of data that

are commonly used for authenticating an individual, as opposed to those that violate privacy, that is, reveal some sensitive information about an individual. This crucial distinction is often overlooked by designers of privacy protection technologies.

The second legal context in which the term “personally identifiable information” appears is privacy law. In the U.S., the Privacy Act of 1974 regulates the collection of personal information by government agencies. There is no overarching federal law regulating private entities, but some states have their own laws, such as California’s Online Privacy Protection Act of 2003. Generic privacy laws in other countries include Canada’s Personal Information Protection and Electronic Documents Act (PIPEDA) and Directive 95/46/EC of the European Parliament, commonly known as the Data Protection Directive.

Privacy laws define PII in a much broader way. They account for the possibility of deductive disclosure and—unlike breach-notification laws—do not lay down a list of informational



attributes that constitute PII. For example, the Data Protection Directive defines personal data as: “any information relating to an [...] natural person [...] who can be identified, directly or indirectly, in particular by reference [...] to one or more factors specific to his physical, physiological, mental, economic, cultural, or social identity.”

The Directive goes on to say that “account should be taken of all the means likely reasonably to be used either by the controller<sup>a</sup> or by any other person to identify the said person.” Similarly, the HIPAA Privacy Rule defines individually identifiable health information as information “(1) That identifies the individual; or 2) With respect to which there is a reasonable basis to believe the information can be used to identify the individual.” What is “reasonable”? This is left open to interpretation by case law. We are not aware of any court decisions that define identifiability in the context of

a The individual or organization responsible for the safekeeping of personal information.

HIPAA.<sup>b</sup> The “safe harbor” provision of the Privacy Rule enumerates 18 specific identifiers that must be removed prior to data release, but the list is not intended to be comprehensive.

### PII and Privacy Protection Technologies

Many companies that collect personal information, including social networks, retailers, and service providers, assure customers that their information will be released only in a “non-personally identifiable” form. The underlying assumption is that “personally identifiable information” is a fixed set of attributes such as names and contact information. Once data records have been “de-identified,” they magically become safe to release, with no way of linking them back to individuals.

The natural approach to privacy pro-

b When the Supreme Court of Iceland struck down an act authorizing a centralized database of “non-personally identifiable” health data, its ruling included factors such as education, profession, and specification of a particular medical condition as part of “identifiability.”

tection is to consider both the data and its proposed use(s) and to ask: What risk does an individual face if her data is used in a particular way? Unfortunately, existing privacy technologies such as *k*-anonymity<sup>6</sup> focus instead on the data alone. Motivated by an attack in which hospital discharge records were re-identified by joining<sup>c</sup> them via common demographic attributes with a public voter database,<sup>5</sup> these methods aim to make joins with external datasets harder by anonymizing the identifying attributes. They fundamentally rely on the fallacious distinction between “identifying” and “non-identifying” attributes. This distinction might have made sense in the context of the original attack, but is increasingly meaningless as the amount and variety of publicly available information about individuals grows exponentially.

To apply *k*-anonymity or its variants such as *l*-diversity, the set of the so-called *quasi-identifier* attributes must be fixed in advance and assumed to

c In the sense of SQL join.



be the same for all users. It typically includes ZIP code, birth date, gender, and/or other demographics. The rest of the attributes are assumed to be non-identifying. De-identification involves modifying the quasi-identifiers to satisfy various syntactic properties, such as “every combination of quasi-identifier values occurring in the dataset must occur at least  $k$  times.”<sup>6</sup> The trouble is that even though joining two datasets on common attributes can lead to re-identification, anonymizing a predefined subset of attributes is not sufficient to prevent it.

### Re-identification without PII

Any information that distinguishes one person from another can be used for re-identifying anonymous data. Examples include the AOL fiasco, in which the content of search queries was used to re-identify a user; our own work, which demonstrated feasibility of large-scale re-identification using movie viewing histories (or, in general, any behavioral or transactional profile<sup>2</sup>) and local structure of social networks;<sup>3</sup> and re-identification based on location information and stylometry (for example, the latter was used to infer the authorship of the 12 disputed Federalist Papers).

Re-identification algorithms are agnostic to the semantics of the data elements. It turns out there is a wide spectrum of human characteristics that enable re-identification: consumption preferences, commercial transactions, Web browsing, search histories, and so forth. Their two key properties are that (1) they are reasonably stable across time and contexts, and (2) the corresponding data attributes are sufficiently numerous and fine-grained that no two people are similar, except with a small probability.

The versatility and power of re-identification algorithms imply that terms such as “personally identifiable” and “quasi-identifier” simply have no technical meaning. While some attributes may be uniquely identifying on their own, *any attribute can be identifying in combination with others*. Consider, for example, the books a person has read or even the clothes in her wardrobe: while no single element is a (quasi)-identifier, any sufficiently large subset uniquely identifies the individual.

Re-identification algorithms based on behavioral attributes must tolerate a certain “fuzziness” or imprecision in attribute values. They are thus more computationally expensive and more difficult to implement than re-identification based on demographic quasi-identifiers. This is not a significant deterrence factor, however, because re-identification is a one-time effort and its cost can be amortized over thousands or even millions of individuals. Further, as Paul Ohm argues, re-identification is “accretive”: the more information about a person is revealed as a consequence of re-identification, the easier it is to identify that person in the future.<sup>4</sup>

### Lessons for Privacy Practitioners

The emergence of powerful re-identification algorithms demonstrates not just a flaw in a specific anonymization technique(s), but the fundamental inadequacy of the entire privacy protection paradigm based on “de-identifying” the data. De-identification provides only a weak form of privacy. It may prevent “peeping” by insiders and keep honest people honest. Unfortunately, advances in the art and science of re-identification, increasing economic incentives for potential attackers, and ready availability of personal information about millions of people (for example, in online social networks) are rapidly rendering it obsolete.

The PII fallacy has important implications for health-care and biomedical datasets. The “safe harbor” provision of the HIPAA Privacy Rule enumerates 18 attributes whose removal and/or modification is sufficient for the data to be considered properly de-identified, with the implication that such data can be released without liability. This appears to contradict our argument that PII is meaningless. The “safe harbor” provision, however, applies only if the releasing entity has “no actual knowledge that the information remaining could be used, alone or in combination, to identify a subject of the information.” As actual experience has shown, any remaining attributes can be used for re-identification, as long as they differ from individual to individual. Therefore, PII has no meaning even in the context of the HIPAA Privacy Rule.

### Beyond De-identification

Developing effective privacy protection technologies is a critical challenge for security and privacy research. While much work remains to be done, some broad trends are becoming clear, as long as we avoid the temptation to find a silver bullet. Differential privacy is a major step in the right direction.<sup>1</sup> Instead of the unattainable goal of “de-identifying” the data, it formally defines what it means for a *computation* to be privacy-preserving. Crucially, it makes no assumptions about the external information available to the adversary. Differential privacy, however, does not offer a universal methodology for data release or collaborative, privacy-preserving computation. This limitation is inevitable: privacy protection has to be built and reasoned about on a case-by-case basis.

Another lesson is that an interactive, query-based approach is generally superior from the privacy perspective to the “release-and-forget” approach. This can be a hard pill to swallow, because the former requires designing a programming interface for queries, budgeting for server resources, performing regular audits, and so forth.

Finally, any system for privacy-preserving computation on sensitive data must be accompanied by strong access control mechanisms and non-technological protection methods such as informed consent and contracts specifying acceptable uses of data. ■

### References

1. Dwork, C. A firm foundation for private data analysis. *Commun. ACM*, (to appear).
2. Narayanan, A. and Shmatikov, V. Robust de-anonymization of large sparse datasets. In *Proceedings of the 2008 IEEE Symposium on Security and Privacy*.
3. Narayanan, A. and Shmatikov, V. De-anonymizing social networks. In *Proceedings of the 2009 IEEE Symposium on Security and Privacy*.
4. Ohm, P. Broken promises of privacy: Responding to the surprising failure of anonymization. *57 UCLA Law Review* 57, 2010 (to appear).
5. Sweeney, L. Weaving technology and policy together to maintain confidentiality. *J. of Law, Medicine, and Ethics* 25 (1997).
6. Sweeney, L. Achieving  $k$ -anonymity privacy protection using generalization and suppression. *International Journal on Uncertainty, Fuzziness, and Knowledge-Based Systems* 10 (2002).

Arvind Narayanan (arvindn@cs.utexas.edu) is a postdoctoral fellow at Stanford University. Vitaly Shmatikov (shmat@cs.utexas.edu) is an associate professor of computer science at the University of Texas at Austin. Their paper on de-anonymization of large sparse datasets<sup>2</sup> received the 2008 PET Award for Outstanding Research in Privacy Enhancing Technologies.

Copyright held by author.



## Inside Risks

# Privacy By Design: Moving from Art to Practice

*Designing privacy into systems at the beginning of the development process necessitates the effective translation of privacy principles, models, and mechanisms into system requirements.*

**M**OST PEOPLE INVOLVED with system development are well aware of the adage that you are better off designing in security and privacy (and pretty much any other “nonfunctional” requirements) from the start, rather than trying to add them later. Yet, if this is the conventional wisdom, why is the conventional outcome so frequently systems with major flaws in these areas?

Part of the problem is that while people know how to talk about functionality, they are typically a lot less fluent in security and privacy. They may sincerely want security and privacy, but they seldom know how to specify what they seek. Specifying functionality, on the other hand, is a little more straightforward, and thus the system that previously could make only regular coffee in addition to doing word processing will now make espresso too. (Whether this functionality actually meets user needs is another matter.)

### Security and Privacy

The fact that it is often not apparent what security and privacy should look like is indicative of some deeper issues. Security and privacy tend to be articulated at a level of abstraction that often makes their specific manifestations less than obvious, to either customers or system developers.

This is not to say the emperor has no clothes; far from it. There are sub-



**Members of staff are seen demonstrating a new whole-body security scanner at Manchester Airport, Manchester, England, in January 2010. Airline passengers bound for the United States faced a hodgepodge of security measures across Europe and airports did not appear to be following a U.S. request for increased screening of passengers from 14 countries.**

stantial bodies of knowledge for some nonfunctional areas, including security, but figuring out how to translate the abstract principles, models, and mechanisms into comprehensive specific requirements for specific systems operating within specific contexts is seldom straightforward. That translation process is crucial to designing these properties into systems, but it

also tends to be the most problematic activity and the activity for which the least guidance is provided. The sheer complexity of most modern systems compounds the problem.

Security, though, is better positioned than privacy. Privacy—or informational privacy at least—certainly has commonly understood and accepted principles in the form of Fair Informa-



ACM's *interactions* magazine explores critical relationships between experiences, people, and technology, showcasing emerging innovations and industry leaders from around the world across important applications of design thinking and the broadening field of the interaction design. Our readers represent a growing community of practice that is of increasing and vital global importance.

**interactions**  
<http://www.acm.org/subscribe>



tion Practices. It presently doesn't have much else. Models and mechanisms that support privacy are scarce, not generally known, and rarely understood by either customers or developers.

As more things become digitized, informational privacy increasingly covers areas for which Fair Information Practices were never envisioned. Biometrics, physical surveillance, genetics, and behavioral profiling are just a few of the areas that are straining Fair Information Practices to the breaking point. More sophisticated models are emerging for thinking about privacy risk, as represented by the work of scholars such as Helen Nissenbaum and Daniel Solove. However, if not associated with privacy protection mechanisms and supported by translation guidance, the impact of such models is likely to be much less than they deserve.

A recent example is the development and deployment of whole-body imaging (WBI) machines at airports for physical screening of passengers. In their original incarnation, these machines perform what has been dubbed a "virtual strip search" due to the body image that is presented. These machines are currently being deployed at U.S. airports in a way that is arguably compliant with Fair Information Practices. Yet they typically operate in a way that many people find offensive.

The intended purpose certainly is not to collect, use, disclose, and retain naked images of people; it is to detect potentially dangerous items they may be carrying on their persons when screened. Fair Information Practices include minimization of personal information collected, used, disclosed, and retained, consistent with the intended purpose.

This has profound implications for how image data is processed, presented, and stored. It should be processed so at no point does there ever exist an exposed body image that can be viewed or stored. It should be presented in a nonexposed form (for example, a chalk outline or a fully clothed person) with indicators where things have been detected. None of it should be retained beyond the immediate encounter. That almost none of these design elements were originally specified illustrates what too often happens

in the absence of applicable models and mechanisms and their requisite translation, along with principles, into effective requirements.

In this instance, Solove's concept of *exposure* provides the necessary (partial) model. Exposure is a privacy violation that induces feelings of vulnerability and distress in the individual by revealing things we customarily conceal. The potential harm from exposure is not restricted to modesty or dignity. A friend is convinced that her pubescent daughter, who is currently extremely self-conscious about her body, would be quite literally traumatized if forced to undergo WBI. If physical strip searches would raise concern, why not WBI? Real damage—physical as well as psychological—can occur in the context of body image neuroses.

If one recognizes from the outset the range of privacy risks represented by exposure, and the relevance of exposure for WBI, one then stands a chance of effectively moving from principles to requirements. Even then, though, the translation process is not necessarily obvious.

Supposedly, the WBI machines being used by the U.S. Transportation Security Administration are not capable of retaining images when in normal operating mode. (They have this capability when in testing mode, though, so significant residual risk may exist.) Other necessary mechanisms were not originally specified. Some models of WBI are being retrofitted to present a nonexposed image, but the issue of intermediate processing remains. Some models developed after the initial wave apparently implement all the necessary control mechanisms; privacy really was designed in. Why wasn't it designed in from the beginning and across the board? The poor state of practice of privacy by design offers a partial explanation. The state of the art, though, is advancing.

The importance of meaningfully designing privacy into systems at the beginning of the development process, rather than bolting it on at the end (or overlooking it entirely), is being increasingly recognized in some quarters. A number of initiatives and activities are using the rubric of *privacy by design*. In Canada, the Ontario Information and Privacy Com-

missioner's Office has published a number of studies and statements on how privacy can be designed into specific kinds of systems. One example is electronic (RFID-enabled) driver's licenses, for which the inclusion of a built-in on/off switch is advocated, thereby providing individuals with direct, immediate, and dynamic control over whether the personal information embedded in the license can be remotely read or not. Such a mechanism would support several Fair Information Practices, most notably collecting personal information only with the knowledge and consent of the individual. This approach is clearly applicable as well to other kinds of RFID-enabled cards and documents carrying personal information.

Similar efforts have been sponsored by the U.K. Information Commissioner's Office. This work has taken a somewhat more systemic perspective, looking less at the application of privacy by design to specific types of technology and more at how to effectively integrate privacy into the system development life cycle through measures such as privacy impact assessments and 'practical' privacy standards. It also emphasizes the potential role of privacy-enhancing technologies (PETs) that can be integrated with or into other systems. While some of these are oriented toward empowering individuals, others—which might more appropriately be labeled Enterprise PETs—are oriented toward supporting organizational stewardship of personal information.

However, state of the art is state of the art. Supporting the translation of abstract principles, models, and mechanisms into implementable requirements, turning this into a repeatable process, and embedding that process in the system development life cycle is no small matter. Security has been at it a lot longer than privacy, and it is still running into problems. But at least security has a significant repertoire of principles, models, and mechanisms; privacy has not really reached this stage yet.

## Conclusion

So, if privacy by design is still a ways off, and security by design still leaves something to be desired, how do we get there from here? There's little

## Security and privacy tend to be articulated at a level of abstraction that often makes their specific manifestations less than obvious.

doubt that appropriately trained engineers (including security engineers) are key to supporting the effective translation of principles, models, and mechanisms into system requirements. There doesn't yet appear to be such a thing as a privacy engineer; given the relative paucity of models and mechanisms, that's not too surprising. Until we build up the latter, we won't have a sufficient basis for the former. For privacy by design to extend beyond a small circle of advocates and experts and become the state of practice, we'll need both.

This will require recognition that there is a distinct and necessary technical discipline of privacy, just as there is a distinct and necessary technical discipline of security—even if neither is fully formed. If that can be accomplished, it will create a home and an incentive for the models and mechanisms privacy by design so badly needs.

This is not to minimize the difficulty of more effectively and consistently translating security's body of knowledge (which is still incomplete) into implementable and robust requirements. Both security and privacy need to receive more explicit and directed attention than they often do as areas of research and education.

Security by design and privacy by design can be achieved only by design. We need a firmer grasp of the obvious. □

**Stuart S. Shapiro** ([s\\_shapiro@acm.org](mailto:s_shapiro@acm.org)) is Principal Information Privacy and Security Engineer at The MITRE Corporation, Bedford MA.

Copyright held by author.

# Calendar of Events

**June 15**

**MobileCloud Workshop**  
(co-located with MobiSys 2010),  
San Francisco, CA,  
Contact: Li Erran Li,  
Email: [erranli@research.bell-labs.com](mailto:erranli@research.bell-labs.com)

**June 15-18**

**International Conference on Informatics in Control, Automation and Robotics**,  
Funchal, Portugal,  
Contact: Joaquim Filipe,  
Email: [jfilipe@insticc.org](mailto:jfilipe@insticc.org)

**June 15-18**

**Computers, Freedom, and Privacy**,  
San Jose, CA,  
Contact: Jon Pincus,  
Email: [jon@achangeiscoming.net](mailto:jon@achangeiscoming.net)

**June 15-18**

**Annual NASA/ESA Adaptive Hardware and Systems Conference**,  
Anaheim, CA,  
Contact: Arslan Tughrul,  
Email: [t.arslan@ed.ac.uk](mailto:t.arslan@ed.ac.uk)

**June 16-18**

**Conference on the Future of the Internet 2010**,  
Seoul Republic of Korea,  
Contact: Dongman Lee,  
Email: [dlee@cs.kaist.ac.kr](mailto:dlee@cs.kaist.ac.kr)

**June 17-18**

**Third International Workshop on Future Multimedia Networking**,  
Krakow, Poland,  
Contact: Mauthe Andreas,  
Email: [a.mauthe@lancaster.ac.uk](mailto:a.mauthe@lancaster.ac.uk)

**June 19-23**

**ACM SIGCHI Symposium on Engineering Interactive Computing Systems**,  
Berlin, Germany,  
Contact: Jean Vanderdonckt  
Email: [jean.vanderdonckt@uclouvain.be](mailto:jean.vanderdonckt@uclouvain.be)

**June 19-23**

**The 37<sup>th</sup> Annual International Symposium on Computer Architecture**,  
Saint-Malo, France,  
Contact: Andre Seznec  
Email: [sez nec@irisa.fr](mailto:sez nec@irisa.fr)



## The Profession of IT

# The Resurgence of Parallelism

*Parallel computation is making a comeback after a quarter century of neglect. Past research can be put to quick use today.*

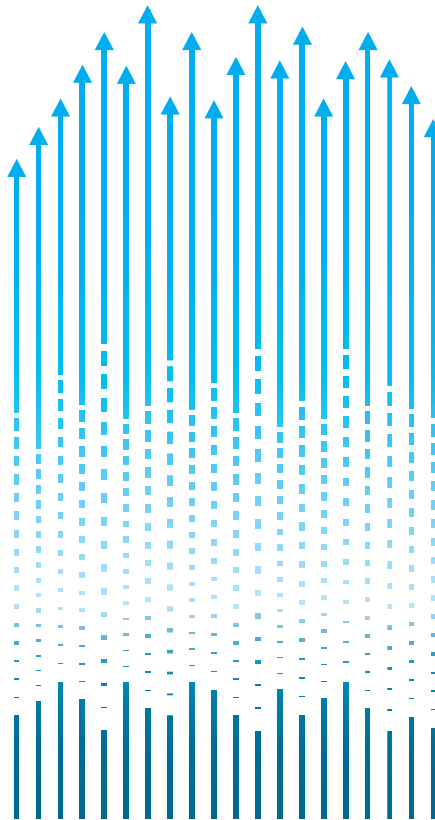
**M**ULTI-CORE CHIPS ARE a new paradigm!" "We are entering the age of parallelism!" These are today's faddish rallying cries for new lines of research and commercial development. Is this really the first time when computing professionals seriously engaged with parallel computation? Is parallelism new? Is parallelism a new *paradigm*?

### Déjà Vu All Over Again

Parallel computation has always been a means to satisfy our never-ending hunger for ever-faster and ever-cheaper computation.<sup>4</sup> In the 1960s and 1970s, parallel computation was extensively researched as a means to high-performance computing. But the commercial world stuck with a quest for faster CPUs and, assisted by Moore's Law, made it to the 2000s without having to seriously engage with parallel computation except for supercomputers. The parallel architecture research of the 1960s and 1970s solved many problems that are being encountered today. Our objective in this column is to recall the most important of these results and urge their resurrection.

### Shared Memory Multiprocessing

The very first multiprocessor architecture was the Burroughs B5000, designed beginning in 1961 by a team led by Robert Barton. It was followed by the B5500 and B6700, along with a defense version, the D850. The architec-



ture survives today in the reverse polish notation HP calculators and in the UniSys ClearPath MCP machines.

Those machines used shared memory multiprocessors in which a crossbar switch connected groups of four processors and memory boxes. The operating system, known as Automatic Scheduling and Operating Program (ASOP), included many innovations. Its working storage was organized as a stack machine. All its

code was "reentrant," meaning that multiple processors could execute the same code simultaneously while computing on separate stacks. The instruction set, which was attuned to the Algol language, was very simple and efficient even by today's RISC standards. A newly spawned process's stack was linked to its parent's stack, giving rise to a runtime structure called "cactus stack." The data memory outside of the stacks was laid out in *segments*; a segment was a contiguous sequence of locations with base and bound defined by a *descriptor*. Segments were moved automatically up and down the memory hierarchy, an early form of virtual memory not based on paging. Elliot Organick's masterful descriptions of these machines make for refreshing and worthwhile reading today.<sup>9,12</sup>

The Burroughs systems were an important influence on research seeking efficient and reliable parallel program structures. A group of researchers at Brown University and General Electric Research Laboratories produced a set of reports on a "contour model" of nested multitask computations in 1971.<sup>12</sup> Those reports give a remarkably clear picture of a parallel programming runtime environment that would suit today's languages well and would resolve many contemporary problems considered as "research challenges." It is a tragedy these ideas have disappeared from the curriculum.

The Burroughs machines disappeared not because of any defect in their architecture, but because of IBM's massive success in marketing the 360 series systems. Moreover, in a process reminiscent of Clayton Christensen's *Innovator's Dilemma*, the low-end assembler-language mini-computers, originally designed to run laboratory instruments, grew up into the minicomputer and then micro-computer market, untainted by any notions of parallel programming.

With the introduction of RISC architectures in the early 1980s, much of the research for high-performance computers was rechanneled toward exploiting RISC for fast chips. It looked at the time that sophisticated compilers could make up for missing functions in the chips.

With two notable exceptions, most of the projects exploring alternatives to the "von Neumann architecture" expired and were not replaced with new projects or other initiatives. One exception was Arvind's Monsoon Project at MIT,<sup>10</sup> which demonstrated that massive parallelism is readily identified in the functional programming language Haskell, and then readily mapped to a shared memory multiprocessor. (Functional languages generate all their values by evaluating functions without side effects.)

The other project involved a group at the Lawrence Livermore National Laboratory studying scientific codes in the functional language Sisal, a derivative of MIT's Val language; Sisal programs were as efficient as Fortran programs and could be readily compiled to massively parallel shared memory supercomputers.<sup>1,2,11</sup>

The current generations of super-computers (and data warehouses) are based on thousands of CPU chips running in parallel. Unlike the innovative designs of the Burroughs systems, their hardware architectures conform to the conventional von Neumann machine. Their operating systems are little more than simple schedulers and message passing protocols, with complex functions relegated to applications running on separate host machines.

The point is clear: ideas for arranging multiple processors to work together in an integrated system have been with us for 50 years. What's new?

## The parallel architecture research of the 1960s and 1970s solved many problems that are being encountered today.

### Determinate Computation

One of the holy grails of research in parallel computation in the 1960s and 1970s was called "determinacy."<sup>7</sup> Determinacy requires that a network of parallel tasks in shared memory always produces the same output for given input regardless of the speeds of the tasks. It should not be confused with a similar word, "deterministic," which would require that the tasks be ordered in the same sequence every time the system runs.

A major result of this research was the "determinacy theorem." A task is a basic computation that implements a function from its inputs to outputs. Two tasks are said to be in conflict if either of them writes into memory cells used by the other. In a system of concurrent tasks, race conditions may be present that make the final output depend on the relative speeds or orders of task execution. Determinacy is ensured if the system is constrained so that every pair of conflicting tasks is performed in the same order in every run of the system. Then no data races are possible. Note that atomicity and mutual exclusion are not sufficient for determinacy: they ensure only that conflicting tasks are not concurrent, but not that they always executed in the same order.

A corollary of the determinacy theorem is that the entire sequence of values written into each and every memory cell during any run of the system is the same for the given input. This corollary also tells us that any system of blocking tasks that communicates by messages using FIFO queues (instead of shared memory) is automatically

determinate because the message queues always present the data items in the same order to the tasks receiving them.

Another corollary is that an implementation of a functional programming language using concurrent tasks is determinate because the functions provide their data privately to their successors when they fire. There is no interference among the memory cells used to transmit data between functions.

Determinacy is really important in parallel computation. It tells us we can unleash the full parallelism of a computational method without worrying whether any timing errors or race conditions will negatively affect the results.

### Functional Programming and Composability

Another holy grail for parallel system has been modular composability. This would mean that any parallel program can be used, without change, as a component of a larger parallel program.

Three principles are needed to enable parallel program composability. David Parnas wrote about two: information hiding and context independence. Information hiding means a task's internal memory cannot be read or written by any other task. Context independence means no part of a task can depend on values outside the task's internal memory or input-output memory. The third principle is argument noninterference; it says that a data object presented as input to two concurrent modules cannot be modified by either.

Functional programming languages automatically satisfy these three principles; their modules are thus composable.

It is an open question how to structure composable parallel program modules from different frameworks when the modules implement non-determinate behavior. Transaction systems are an extreme case. Because their parallel tasks may interfere in the records they access, they use locking protocols to guarantee mutual exclusion. Transaction tasks cannot be ordered by a fixed order—their non-determinacy is integral to their function. For example, an airplane seat goes to whichever task requested it first. The

problem is to find a way to reap the benefits of composability for systems that are necessarily nondeterminate.

### Virtual Memory

There are obvious advantages if the shared memory of a parallel multi-processor could be a virtual memory. Parameters can be passed as pointers (virtual addresses) without copying (potentially large) objects. Compilation and programming are greatly simplified because neither compilers nor programmers need to manage the placement of shared objects in the memory hierarchy of the system; that is done automatically by the virtual memory system.

Virtual memory is essential for modular composability when modules can share objects. Any module that manages the placement of a shared object in memory violates the information hiding principle because other modules must consult it before using a shared object. By hiding object locations from modules, virtual memory enables composability of parallel program modules.

There are two concerns about large virtual memory. One is that the virtual addresses must be large so that they encompass the entire address space in which the large computations proceed. The Multics system demonstrated that a very large virtual address space—capable of encompassing the entire file system—could be implemented efficiently.<sup>8</sup> Capability-based addressing<sup>5,6</sup> can be used to implement a very large address space.

The other concern about large virtual memory pertains to performance. The locality principle assures us that

**Parallelism is not new; the realization that it is essential for continued progress in high-performance computing is.**

each task accesses a limited but dynamically evolving working set of data objects.<sup>3</sup> The working set is easily detected—it is the objects used in a recent backward-looking window—and loaded into a processor's cache. There is no reason to be concerned about performance loss due to an inability to load every task's working set into its cache.

What about cache consistency? A copy of a shared object will be present in each sharing task's cache. How do changes made by one get transmitted to the other? It would seem that this problem is exacerbated in a highly parallel system because of the large number of processors and caches.

Here again, the research that was conducted during the 1970s provides an answer. We can completely avoid the cache consistency problem by never writing to shared data. That can be accomplished by building the memory as a write-once memory: when a process writes into a shared object, the system automatically creates a copy and tags it as the current version. These value sequences are unique in a determinate system. Determinate systems, therefore, give a means to completely avoid the cache consistency problem and successfully run a very large virtual memory.

### Research Challenges

Functional programming languages (such as Haskell and Sisal) currently support the expression of large classes of application codes. They guarantee determinacy and support composability. Extending these languages to include stream data types would bring hazard-free expression to computations involving inter-module pipelines and signal processing. We badly need a further extension to support programming in the popular object-oriented style while guaranteeing determinacy and composability.

We have means to express nondeterminate computation in self-contained environments such as interactive editors, version control systems, and transaction systems. We sorely need approaches that can combine determinate and nondeterminate components into well-structured larger modules.

The full benefits of functional programming and composability cannot

be fully realized unless memory management and thread scheduling are freely managed at runtime. In the long run, this will require merging computational memory and file systems into a single, global virtual memory.

### Conclusion

We can now answer our original questions. Parallelism is not new; the realization that it is essential for continued progress in high-performance computing is. Parallelism is not yet a paradigm, but may become so if enough people adopt it as the standard practice and standard way of thinking about computation.

The new era of research in parallel processing can benefit from the results of the extensive research in the 1960s and 1970s, avoiding rediscovery of ideas already documented in the literature: shared memory multiprocessing, determinacy, functional programming, and virtual memory. □

### References

1. Cann, D. Retire Fortran?: A debate rekindled. *Commun. ACM* 35, 8 (Aug. 1992), 81–89.
2. Cann, D. and Feo, J. SISAL versus FORTRAN: A comparison using the Livermore loops. In *Proceedings of the 1990 ACM/IEEE Conference on Supercomputing*. IEEE Computer Society Press, 1990.
3. Denning, P. Virtual memory. *ACM Computing Surveys* 2, 3 (Sept. 1970), 153–189.
4. Denning, P. and Tichy, W. Highly parallel computation. *Science* 250 (Nov. 30, 1990), 1217–1222.
5. Dennis, J. and Van Horn, E.C. Programming semantics for multi-programmed computations. *Commun. ACM* 9, 3 (Mar. 1966), 143–155.
6. Fabry, R. Capability-based addressing. *Commun. ACM* 17, 7 (July 1974), 403–412.
7. Karp, R.M. and Miller, R.E. Properties of a model for parallel computations: Determinacy, termination, queueing. *SIAM Journal of Applied Mathematics* 14, 6 (Nov. 1966), 1390–1411.
8. Organick, E.I. *The Multics System: An Examination of Its Structure*. MIT Press, 1972.
9. Organick, E.I. Computer systems organization: The B5700/B6700. *ACM Monograph Series*, 1973. LCN: 72-88334.
10. Papadopoulos, G.M. and Culler, D.E. Monsoon: An explicit token-store architecture. In *Proceedings of the 17th Annual International Symposium on Computer Architecture*. (1990), 82–91.
11. Sarkar, V. and Cann, D. POSC—A partitioning and optimizing SISAL compiler. In *Proceedings of the 4th International Conference on Supercomputing*. IEEE Computer Society Press, 1990.
12. Tou, J. and Wegner, P., Eds. Data structures in programming languages. *ACM SIGPLAN Notices* 6 (Feb. 1971), 171–190. See especially papers by Wegner, Johnston, Berry, and Organick.

**Peter J. Denning** (pjd@nps.edu) is the director of the Cebrowski Institute for Innovation and Information Superiority at the Naval Postgraduate School in Monterey, CA, and is a past president of ACM.

**Jack B. Dennis** (dennis@csail.mit.edu) is Professor Emeritus of Computer Science and Engineering at MIT and a Principal Investigator in the Computer Science and Artificial Intelligence Laboratory. He is an Eckert-Mauchly Award recipient and a member of the NAE.

Copyright held by author.





## Kode Vicious Plotting Away

*Tips and tricks for visualizing large data sets.*

### Dear KV,

I've been working with some code that generates massive data sets, and while I'm perfectly happy looking at the raw textual output to find patterns, I'm finding that more and more often I have to explain my data to people who are either unwilling to or incapable of understanding the data in a raw format. I'm now being required to generate summaries, reports, and graphs for these people, and, as you can imagine, they are the ones with control over the money in the company, so I also have to be nice to them. I know this isn't exactly a coding question, but what do you do when you have to take the bits you understand quite well and summarize them for people like this?

### If It Ain't Text...

### Dear Text,

Since I often come across as some sort of hard-core, low-level, bits-and-bytes kind of guy, I gather that you're assuming my answer will be to tell management—and from your description these people must be management—to take their fancy graphs and, well, do something that would give them paper cuts in hard-to-reach places. Much as I like to give just that kind of advice, the fact is, it's just as important to be able to transform large data sets from columns and lines of numbers into something that is a bit more compact and still as descriptive. For the polite and well-written version of this type of advice, please see the classic work by Ed-



ward Tufte, *The Visual Display of Quantitative Information*. Now for the Kode Vicious Kwik Course on Visualization, please read on.

While I agree it is cool to be able to look at some incredibly confusing output in text and be able to pick out the needle you're looking for, and while I'm sure this impresses many of your coder friends, this is just not a skill that's going to take you very far. I also find that programmers who cannot understand the value in a single-page graph of their results are the same kinds of programmers who should not

be allowed to code on their own.

One should approach any such problem as a science experiment, and scientists know how to represent their results in many ways, including plotting them on paper. At some point in your career you're going to have to figure out how to get easy-to-read results that you can look at and compare side by side. A plot of your data can, when done well, give you a lot of information and tell you a lot about what might be happening with your system. Note the all-important phrase, when done well, in that previous sentence. As is the case

with many tools, the plotting of data can mislead you as easily as it can lead you somewhere.

There are plenty of tools with which to plot your data, and I usually shy away from advocating particular tools in these responses, but I can say that if you were trying to plot *a lot* of data, where a lot is more than 32,767 elements, you would be wise to use something like gnuplot. Every time I've seen people try to use a certain vendor's spreadsheet to plot data sets larger than 32,767, things have gone awry—I might even say that they were brought up “short” by that particular program. The advantage of gnuplot is that as long as you have a lot of memory (and memory is inexpensive now), you can plot very large data sets. KV recently outfitted a machine with 24GB of RAM just to plot some important data. I'm a big believer in big memory for data, but not for programs, but let's just stop that digression here.

Let's now walk through the important points to remember when plotting data. The first is that if you intend to compare several plots, your measurement axis—the one on which you're showing the magnitude of a value—absolutely must remain constant or be easily comparable among the total set of graphs that you generate. A plot with a *y*-axis that goes from 0 to 10 and another with a *y*-axis from 0 to 25 may *look* the same, but their meaning is completely different. If the data you're plotting runs from 0 to 25, then all of your graphs should run from, for example, 0 to 30. Why would you waste those last five ticks? Because when you're generating data from a large data set, you might have missed something, perhaps a crazy outlier that goes to 60, but only on every 1,000th sample. If you set the limits of your axes too tightly initially, then you might never find those outliers, and you would have done an awful lot of work to convince yourself—and whoever else sees your pretty little plot—that there really isn't a problem, when in fact it was right under your nose, or more correctly, right above the limit of your graph.

Since you mention you are plotting large data sets, I'll assume you mean more than 100,000 points. I have routinely plotted data that runs into the millions of individual points. When you plot the data the first time, it's im-

portant not only to get the *y*-axis limits correct, but also to plot as much data as absolutely possible, given the limits of the system on which you're plotting the data. Some problems or effects are not easily seen if you reduce the data too much. Reduce the data set by 90% (look at every 10th sample), and you might miss something subtle but important. If your data won't all fit into main memory in one go, then break it down by chunks along the *x*-axis. If you have one million samples, graph them 100,000 at a time, print out the graphs, and tape them together. Yes, it's kind of a quick-and-dirty solution but it works, trust me.

Another problem occurs when you want to compare two data sets directly on the same plot. Perhaps you have data from several days and you want to see how Wednesday and Thursday compare, but you don't have enough memory to plot both days at once, only enough for one day at a time. You could beg your IT department for more memory or, if you have a screwdriver, “borrow” some memory from a coworker, but such measures are unnecessary if you have a window. Print both data sets, making sure both axes line up, and then hold the pages up to the window. Oh, when I said “window,” I meant one that allows light from that bright yellow ball in the sky to enter your office, not one that is generated by your computer.

Thus far I have not mentioned the *x*-axis, but let's remedy that now. If you're plotting data that changes over time, then your *x*-axis is actually a time axis. The programmers who label this “samples,” and then do all kinds of internal mental transformations, are legion—and completely misguided. While you might know that your samples were taken at 1KHz and therefore that every 1,000 samples is one second, and 360,000 samples is an hour, most of the people who see your plots are not going to know this, even if you cleverly label your *x*-axis “1KHz.” If you're plotting something against time, then your *x*-axis really should be time.

This recommendation is even more important when graphing long-running data—for example, a full working day. It turns out that computers are slaves to people and while many people have predicted that the work done by computers would be far more consis-

tent over a 24-hour day than work done by humans, all of those people have been, and continue to be, dead wrong. If you're plotting data over a day, then it is highly likely that you will see changes when people wake up, when they go to work, take meals, go home, and sleep. It might be vitally important for you to notice that something happens every day at 4 P.M. Perhaps your systems in England are recording when people take tea, rather than an odd slowdown in the system. The system you're watching might be underutilized because the tea trolley just arrived! If your plot has time, then use time as an axis.

As I wrap this up, you may have noticed that I did not mention color, fonts, font size, or anything else related to how the graph looks on paper. I didn't leave these factors out because I'm a total nerd who can't match any of his own clothes. I can easily match clothes, since black goes with everything. Most people I've seen generating graphs spend far too much time picking a color or a font. Take the defaults; just make sure the lines on the graph are consistently representing your data. Choosing a graph color or text font before getting the data correctly plotted is like spending hours twiddling your code highlighting colors in your IDE instead of doing the actual work of coding. It's a time waster. Now, get back to work.

**KV**

#### Related articles on [queue.acm.org](http://queue.acm.org)

##### Code Spelunking Redux

George V. Neville-Neil

<http://queue.acm.org/detail.cfm?id=1483108>

##### Unifying Biological Image Formats with HDF5

Matthew T. Dougherty, Michael J. Folk,

Erez Zadok, Herbert J. Bernstein,

Frances C. Bernstein, Kevin W. Eliceiri,

Werner Benger, Christoph Best

<http://queue.acm.org/detail.cfm?id=1628215>

##### A Conversation with Jeff Heer, Martin Wattenberg, and Fernanda Viégas

<http://queue.acm.org/detail.cfm?id=1744741>

George V. Neville-Neil ([kv@acm.org](mailto:kv@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.

## Law and Technology

# Intel's Rebates: Above Board or Below the Belt?

*Over several years, Intel paid billions of dollars to its customers. Was it to force them to boycott products developed by its rival AMD or so they could sell its microprocessors at lower prices?*

**O**VER A FIVE-YEAR period, Dell allegedly received payments from Intel averaging \$300 million per quarter.<sup>a</sup> The Attorney General of the State of New York, Andrew M. Cuomo, accuses the Santa Clara-based chip firm of making these payments to force its OEM customer not to use AMD's x86 CPUs in its computers, in violation of antitrust law. Intel is alleged to have infringed Section 2 of the Sherman Act, which opposes behavior by firms aimed at creating or preserving a monopoly other than by merit. In December 2009, the Federal Trade Commission also filed suit against Intel.<sup>b</sup> The FTC accuses the chip maker of numerous anticompetitive unfair practices, including various payments to its customers in the form of lump sums or discounts.

In Europe, the case is closed, or almost. The billions of dollars that Intel paid to Dell, HP, and several other firms were deemed anticompetitive behavior. The European Commission found that the payments amounted to a strategy to exclude AMD from the microprocessor market. They were considered akin to rebates and re-



**A netbook equipped with an Intel Atom processor is demonstrated at the International Consumer Electronics Show in Las Vegas in 2009.**

strictions imposed on buyers, which are incompatible with European antitrust law. The Commission found against Intel in May 2009 and fined the firm almost \$2 billion.<sup>c</sup> So, instead of going to its customers, Intel's money replenished the coffers of the European Union! Intel immediately appealed to the EU Court of First Instance in Luxembourg. It also signed a \$1.25 billion settlement agreement with Dell to put an end to its antitrust

and patent allegations.

Intel considers the payments it made to customers were, on the contrary, a reflection of vigorous competition and beneficial to consumers.

Who's right? Who's wrong? The parties offer diametrically opposed versions of the story.

### Plaintiff Perspective

The story told by the plaintiff State of New York and by the European Commission can be summed up as follows. Intel and AMD are practically the only manufacturers of x86 CPUs, the micro-

a Complaint by Attorney General of the State of New York, Andrew M. Cuomo against Intel Corporation before the United States District Court for the District of Delaware, November 4, 2009.

b Administrative complaint of the U.S. FTC against Intel Corporation, docket No. 9341, December 14, 2009.

c European Commission's decision against Intel Corp., case COMP/37.990, May 13, 2009.



processors inside most computers. Although four times the size of AMD, Intel was outpaced by the smaller firm in innovation. In particular, Intel is having more trouble negotiating the transition from 32-bit architecture to the 64-bit architecture that makes computers more powerful. According to New York Attorney General Cuomo, Intel has a “big competitive hole in its product development roadmap.” In 2003, AMD was the first to introduce a new-generation processor for the high-end,

**Who's right?  
Who's wrong?  
The parties offer  
diametrically  
opposed versions  
of the story.**

high-margin corporate server market. Intel feared its competitor would erode its profits on this segment, since business users would be eager to purchase AMD-based desktops and notebooks.

To prevent that market shift, Intel paid Dell and HP to purchase Intel microprocessors almost exclusively, and paid Acer and Lenovo to delay the launch of their AMD-based notebooks. In other words, Intel paid its customers to protect a segment of its market. Dell was by far the biggest beneficiary of these practices. Between February 2002 and January 2007, Dell received more than \$6 billion in return for maintaining an exclusive procurement agreement with Intel. Without these payments, Dell would have reported a loss in some quarters. According to the State of New York, the Federal Trade Commission, and the European Commission, the money that Intel paid its customers was conditional on their boycotting AMD's products. In technical terms, the retroactive rebates given to some OEM customers are loyalty rebates, and the restrictions imposed on OEMs' sales policies are naked re-

strictions. In Europe, both are generally prohibited because they are perceived to be anticompetitive because they tend to exclude competitors and reduce consumer choice.

#### **Defendant Perspective**

Intel's version is nothing like the previous story.<sup>d</sup> Since 2000, the Santa Clara-based chip maker has faced aggressive price competition from AMD and it has responded by defending itself fairly. AMD's failure to succeed in some market segments is due to its own shortcomings, especially insufficient production capacity, not to any action by Intel. Between 2002 and 2007, the price of microprocessors fell by 36% on average per year and AMD's market share among the main computer manufacturers has risen from 8% to 22%. These figures contradict the claims that Intel has behaved

<sup>d</sup> See “Why the European Commission's Intel decision is Wrong,” and “Intel's Response to the EC's Provisional Non-Confidential Version of the Commission Decision of 13 May 2009,” September 21, 2009; <http://www.intel.com/pressroom/legal/news.htm>

## Announcing ACM's Newly Improved Career & Job Center!

*Are you looking for your next IT job? Do you need Career Advice?*  
**Visit ACM's newly enhanced career resource at:**  
<http://www.acm.org/careercenter>



**The ACM Career & Job Center offers ACM members a host of benefits including:**

- A highly targeted focus on job opportunities in the computing industry
- Access to hundreds of corporate job postings
- Resume posting keeping you connected to the employment market while letting you maintain full control over your confidential information
- An advanced Job Alert system that notifies you of new opportunities matching your criteria
- Career coaching and guidance from trained experts dedicated to your success
- A content library of the best career articles compiled from hundreds of sources, and much more!

**The ACM Career & Job Center is the perfect place to  
begin searching for your next employment opportunity!**

**Visit today at <http://www.acm.org/careercenter>**



Association for  
Computing Machinery

*Advancing Computing as a Science & Profession*

like a monopolist and AMD has been squeezed out of the market. Computer manufacturers know how to play off their two suppliers to their advantage. Intel claims that the payments to customers were not tied to exclusive or near-exclusive purchasing commitments, but were volume-based discounts enabled by economies of scale in production. Thanks to Intel's discount policy, consumers benefit from lower prices. The prices were always higher than Intel's costs. Therefore Intel cannot be accused of selling below cost to drive out AMD.

Whose story should we believe? How can we tell who's right?

In order to decide between the two versions, the first thing to do is of course to look at the facts. This is not easy for an outside observer (including this writer) because the evidence is off limits. Only the public statements and official decisions are available. In the U.S. lawsuits, the State of New York's complaint and the FTC's statements total less than 100 pages. In the EU case, the material is more abundant. The European Commission's decision against Intel runs to more than 500 pages and cites numerous statements by the parties. For example, a Lenovo purchasing manager wrote in an email message dated December 11, 2006, "Last week Lenovo cut a lucrative deal with Intel. As a result of this, we will not be introducing AMD-based products in 2007 for our notebook products." Thousands of figures are also reported. Unfortunately, in order to respect business secrecy, almost all the figures have been deleted from the public version of the decision. Thus, there is no information about the amount of the discounts granted to Dell.

A factual approach is also hampered by the absence of formal contracts. What Intel requested in exchange for the payments to its customers is not mentioned in written documents. Most of the agreements were oral and sealed with a handshake. The few written agreements raise no antitrust concerns. The State of New York and the European Commission accuse Intel of having deliberately removed from the written documents the litigious clauses with respect to antitrust law. If the allegations were proved true, the antitrust agencies would be dealing

## The problem of access to the data and evidence makes it impossible to verify the validity of the different viewpoints.

with a situation akin to a cartel. Since the agreements were secret, evidence is scant and often only indirect.

For want of being able to decide on the basis of the facts, an outside observer can call theory to the rescue.

The first principle to recall is that antitrust law seeks to protect consumers, not competitors. It does not oppose the elimination of less-efficient competitors; it prohibits behavior of firms that results in higher prices or lower-quality products. While clearly detrimental to AMD, did Intel's actions harm consumers?

In the case of the naked restrictions, the damage to consumers is not in doubt. Let's take the example of Intel's lump sum payments to HP, Lenovo, and Acer in exchange for delaying the launch of their AMD-based computers. That practice (if proven) did hurt consumers: some had to wait before buying the product they preferred, while others, in more of a hurry, had to buy hardware that was not their first choice. Moreover, consumers who were not interested in buying AMD-based desktops and notebooks did not gain anything. The money paid by Intel did not affect the OEMs' marginal cost and, consequently, the price of their computers. Intel and the firms it paid off were the only beneficiaries of these transactions.

The case of the rebates is a more complicated situation. When rebates are linked to volumes purchased, they are good for consumers. They enable manufacturers to pass on some of the savings from economies of scale in production and distribution. In other words, they bring prices down for the consumer. But retroactive rebates tied

to market share targets (for example, the buyer receives a rebate if it covers X% of its requirements with the same supplier) are a different story. If a competitor wants to obtain a significant share of the customer's purchases, it must compensate for the loss of the rebates. For example, if Intel offers \$100,000 on the condition that HP fulfills 95% of its requirements with Intel, AMD will be forced to offer the same amount if it wants HP to buy more than 5% of its chips from AMD. That threshold effect can have surprising effects. It would explain, for example, why HP refused AMD's offer of a million Athlon XP processors free of charge. If the gift is worth less than the rebate forfeited by not purchasing 95% of its requirements from Intel, it is rational for HP to refuse it.

### Conclusion

Economic literature<sup>e</sup> shows this threshold effect can lead to the exclusion of competitors that are at least as efficient as the firm offering the rebates. And consumers lose out on two counts. First, there are no more competitors to push down the price set by the dominant firm. So consumers pay higher prices. Second, there is no competitive pressure driving the firm to innovate. So products are manufactured at higher cost and their quality stagnates.

The European Commission sought to show that Intel's loyalty rebates indeed had a foreclosure effect. According to the Commission, a rival with the same costs as Intel would have been excluded. Intel contests this conclusion by finding fault with the Commission's calculations. But once again, the problem of access to the data and evidence makes it impossible to verify the validity of the different viewpoints. Theory without the facts is unfortunately of little use for vindicating either the defendant Intel or the plaintiffs. ■

<sup>e</sup> See, for example, Nicolas Economides, "Loyalty/Requirement Rebates and the Antitrust Modernization Commission: What is the Appropriate Liability Standard?", *Antitrust Bulletin* 54, 2 (Summer 2009), 259-279.

**François Lévêque** (francois.leveque@mines-paristech.fr) is a professor of law and economics at Mines-ParisTech in Paris, France.

Copyright held by author.

## Viewpoint

# Institutional Review Boards and Your Research

*A proposal for improving the review procedures for research projects that involve human subjects and their associated identifiable private information.*

**R**ESearchers in computer science departments throughout the U.S. are violating federal law and their own organization's regulations regarding human subjects research—and in most cases they don't even know it. The violations are generally minor, but the lack of review leaves many universities open to significant sanctions, up to and including the loss of all federal research dollars. The lack of review also means that potentially hazardous research has been performed without adequate review by those trained in human subject protection.

We argue that much computer science research performed with the Internet today involves human subject data and, as such, must be reviewed by Institutional Review Boards—including nearly all research projects involving network monitoring, email, Facebook, other social networking sites and many Web sites with user-generated content. Failure to address this issue now may cause significant problems for computer science in the near future.

### Prisons and Syphilis

At issue are the National Research Act (NRA) of 1974<sup>a</sup> and the Common Rule,<sup>b</sup>

a PL 93-348, see <http://history.nih.gov/research/downloads/PL93-348.pdf>

b 45 CFR 46, see <http://www.hhs.gov/ohrp/humansubjects/guidance/45cfr46.htm>



which together articulate U.S. policy on the Protection of Human Subjects. This policy was created following a series of highly publicized ethical lapses on the part of U.S. scientists performing federally funded research. The most objectionable cases involved human medical experimentation—specifically the Tuskegee Syphilis Experiment, a 40-year long U.S. government project that deliberately withheld syphilis treatment

from poor rural black men. Another was the 1971 Stanford Prison Experiment, funded by the U.S. Office of Naval Research, in which students playing the role of prisoners were brutalized by other students playing the roles of guards.

The NRA requires any institution receiving federal funds for scientific research to set up an Institutional Review Board (IRB) to approve any use of humans *before* the research takes



place. The regulation that governs these boards is the Common Rule—“Common” because the same rule was passed in 1991 by each of the 17 federal agencies that fund most scientific research in the U.S.

Computer scientists working in the field of Human-Computer Interaction (HCI) have long been familiar with the Common Rule: any research that involves recruiting volunteers, bringing them into a lab and running them through an experiment obviously involves human subjects. NSF grant applications specifically ask if human subjects will be involved in the research and require that applicants indicate the date IRB approval was obtained.

But a growing amount of research in other areas of computer science also involves human subjects. This research doesn’t involve live human beings in the lab, but instead involves network traffic monitoring, email, online surveys, digital information created by humans, photographs of humans that have been posted on the Internet, and human behavior observed via social networking sites.

The Common Rule creates a four-part test that determines whether or not proposed activity must be reviewed by an IRB:

1. The activity must constitute scientific “research,” a term that the Rule broadly defines as “a systematic investigation, including research development, testing and evaluation, designed to develop or contribute to generalizable knowledge.”<sup>c</sup>

2. The research must be federally funded.<sup>d</sup>

3. The research must involve human subjects, defined as “a living individual about whom an investigator (whether professional or student) conducting research obtains (1) data through intervention or interaction with the individual, or (2) identifiable private information.”<sup>e</sup>

4. The research must not be “exempt” under the regulations.<sup>f</sup>

The exemptions are a kind of safety valve to prevent IRB regulations from becoming utterly unworkable. For

## Much computer science research performed with the Internet today involves human subject data and, as such, must be reviewed by Institutional Review Boards.

computer scientists the relevant exemptions are “research to be conducted on educational practices or with educational tests” (§46.101(b)(1&2)); and research involving “existing data, documents, [and] records...” provided that the data set is either “publicly available” or that the subjects “cannot be identified, directly or through identifiers linked to the subjects” (§46.101(b)(4)). Surveys, interviews, and observations of people in public are generally exempt, provided that identifiable information is not collected, and provided that the information collected, if disclosed, could not “place the subjects at risk of criminal or civil liability or be damaging to the subjects’ financial standing, employability, or reputation” (§46.101(b)(2)(i&ii)).

IRBs exist to review proposed research and protect the interests of the human subjects. People *can* participate in dangerous research, but it’s important that people are informed, if possible, of the potential risks and benefits—both to themselves and to society at large.

What this means to computer scientists is that *any* federally funded research involving data generated by people that is “identifiable” and not public probably requires approval in advance by your organization’s IRB. This includes obvious data sources like network traffic, but it also includes not so obvious sources like software that collects usage statistics and “phones home.”

Complicating matters is the fact that the Common Rule allows organizations to add additional requirements. Indeed, many U.S. universities require IRB approval for *any* research involving human subjects, regardless of funding source. Most universities also prohibit researchers from determining if their own research is exempt. Instead, U.S. universities typically require that all research involving human beings be submitted to the school’s IRB.

This means a broad swath of “exempt” research involving publicly available information nevertheless requires IRB approval. Performing social network analysis of Wikipedia pages may fall under IRB purview: Wikipedia tracks which users edited which pages, and when those edits were made. Using Flickr pages as a source of JPEGs for analysis may require IRB approval, because Flickr pages frequently have photos of people (identifiable information), and because the EXIF “tags” that many cameras store in JPEG images may contain serial numbers that can be personally identifiable. Analysis of Facebook poses additional problems and may not even qualify as exempt: not only is the information personally identifiable, but it is frequently not public. Instead, Facebook information is typically only available to those who sign up for the service and get invited into the specific user’s network.

We have spoken with quite a few researchers who believe the IRB regulations do not apply to them because they are working with “anonymized” data. Ironically, the reverse is probably true: IRB approval is required to be sure the data collection is ethical, that the data is adequately protected prior to anonymization, and that the anonymization is sufficient. Most schools do not allow the experimenters to answer these questions for themselves, because doing so creates an inherent conflict of interest. Many of these researchers were in violation of their school’s regulations; some were in violation of federal regulations.

### How to Stop Worrying and Love the IRB

Many IRBs are not well equipped to handle the fast-paced and highly technical nature of computer-related research. Basic questions arise, such as,

c §46.102 (d)

d §46.103 (a)

e §46.102 (f)

f §46.101 (b)

Are Internet Protocol addresses personally identifiable information? What is “public” and what is not? Is encrypted data secure? Can anonymized data be re-identified? Researchers we have spoken with are occasionally rebuffed by their IRBs—the IRBs insist that no humans are involved in the research—ignoring that regulations also apply to “identifiable private information.”

Another mismatch between computer science research and IRBs is timescale. CS research progresses at a much faster pace than research in the biomedical and behavioral fields. In one case we are aware of, an IRB took more than a year to make a decision about a CS application. But even two or three months to make a decision—typical of many IRBs—is too slow for a student in a computer science course who wants to perform a social network analysis as a final project.

For example, one of our studies, which involved observing how members of our university community responded to simulated phishing attacks over a period of several weeks, had to be shortened after being delayed two months by an understaffed IRB. With the delayed start date, part of the study would have taken place over winter break, when few people are on campus. Another study we worked on was delayed three months after an IRB asked university lawyers to review a protocol to determine whether it would violate state wiretap laws.

In another case, researchers at Indiana University worked with their IRB and the school’s network security group to send out phishing attacks based on data gleaned from Facebook.<sup>g</sup> Because of the delays associated with the approval process, the phishing messages were sent out at the end of the semester, just before exams, rather than at the beginning of the semester. Many recipients of the email complained vociferously about the timing.

Another reason computer scientists have problems with IRBs is the level of detail the typical IRB application requires. Computer scientists, for the most part, are not trained to carefully plan out an experiment in advance, to

## It is becoming increasingly easy to collect human subjects data over the Internet that needs to be properly protected to avoid harming subjects.

figure out which data will be collected, and then to collect the results in a manner that protects the privacy of the data subjects. (Arguably, computer scientists would benefit from better training on experimental design, but that is a different issue.) We have observed that many IRB applications are delayed because of a failure on the part of CS researchers to make these points clear.

Finally, many computer scientists are unfamiliar with the IRB process and how it applies to them, and may be reluctant to engage with their IRB after having heard nothing but complaints from colleagues who have had their studies delayed by a slow IRB approval process. While the studies that CS researchers perform are often exempt or extremely low risk, it is becoming increasingly easy to collect human subjects data over the Internet that needs to be properly protected to avoid harming subjects. Likewise, the growing amount of research involving honeypots, botnets, and the behavior of anonymity systems would seem to require IRBs, since the research involves not just software, but humans—both criminals and victims.

The risks to human subjects from computer science research are not always obvious, and the IRB can play an important role in helping computer scientists identify these risks and insure that human subjects are adequately protected. Is there a risk that data collected on computer security incidents could be used by employers to identify underperforming computer security administrators? Is there a risk that ano-

nymized search engine data could be re-identified to reveal what particular individuals are searching for? Can network traffic data collected for research purposes be used to identify copyright violators? Can posts to LiveJournal and Facebook be correlated to learn the identities of children who are frequently left home alone by their parents?

In order to facilitate more rapid IRB review, we recommend the development of a new, streamlined IRB application process. Experimenters would visit a Web site that would serve as a self-serve “IRB kiosk.” This site would ask experimenters a series of questions to determine whether their research qualifies as exempt. These questions would also serve to guide experimenters in thinking through whether their research plan adequately protects human subjects. Qualifying experimenters would receive preliminary approval from the kiosk and would be permitted to begin their experiments. IRB representatives would periodically review these self-serve applications and grant final approval if everything was in order.

Such a kiosk is actually permissible under current regulations, provided that the research is exempt. A kiosk could even be used for research that is “expedited” under the Common Rule, since expedited research can be approved by the IRB Chair or by one or more “experienced reviewers.”<sup>h</sup> In the case of non-exempt expedited research, the results of the Kiosk would be reviewed by such a reviewer prior to permission being given to the researcher.

Institutional Review Board chairs from many institutions have told us informally that they are looking to computer scientists to come up with a workable solution to the difficulty of applying the Common Rule to computer science. It is also quite clear that if we do not come up with a solution, they will be forced to do so. ■

<sup>h</sup> §46.110 (b)

**Simson L. Garfinkel** (slgarfin@nps.edu) is an associate professor at the U.S. Naval Postgraduate School in Monterey, CA.

**Lorrie Faith Cranor** (lorrie+@cs.cmu.edu) is an associate professor of computer science and engineering and public policy and the director of the CyLab Usable Privacy and Security Laboratory at Carnegie Mellon University in Pittsburgh, PA.

Copyright held by author.

<sup>g</sup> T. Jagatic, N. Johnson, M. Jakobsson, and F. Menczer. Social phishing. *Commun. ACM* 50, 10 (Oct. 2007), 94–100.

## Interview

# An Interview with Ed Feigenbaum

*ACM Fellow and A.M. Turing Award recipient Edward A. Feigenbaum, a pioneer in the field of expert systems, reflects on his career.*

**T**HE COMPUTER HISTORY Museum has an active program to gather videotaped histories from people who have done pioneering work in this first century of the information age. These tapes are a rich aggregation of stories that are preserved in the collection, transcribed, and made available on the Web to researchers, students, and anyone curious about how invention happens. The oral histories are conversations about people's lives. We want to know about their upbringing, their families, their education, and their jobs. But above all, we want to know how they came to the passion and creativity that leads to innovation.

Presented here are excerpts<sup>a</sup> from four interviews with Edward A. Feigenbaum, the Kumagai Professor of Computer Science, Emeritus, at Stanford University and a pioneering researcher in artificial intelligence. The interviews were conducted in 2007 separately by Donald Knuth and Nils Nilsson, both professors of computer science at Stanford University. —Len Shustek

### What was your family background?

I was born in New Jersey in 1936 to a culturally Jewish family. That Jewish culture thinks of itself as the people of

<sup>a</sup> Oral histories are not scripted, and a transcript of casual speech is very different from what one would write. I have taken the liberty of editing liberally and reordering freely for presentation. For the original transcripts, see <http://archive.computerhistory.org/search/oh/>



the book, and so there's a tremendous focus on learning, and books, and reading. I learned to read very early.

### What got you interested in science and engineering?

My stepfather was the only one in the family who had any college education. Once a month he would take me to the Hayden Planetarium of the American Museum of Natural History. I got really interested in science, mostly through astronomy, at about 10 years old.

My stepfather worked as an accountant and had a Monroe calculator. I was absolutely fascinated by these cal-

culators and learned to use them with great facility. That was one of my great skills—in contrast to other friends of mine whose great skills were things like being on the tennis team.

I was a science kid. I would read *Scientific American* every month—if I could get it at the library. One book that really sucked me into science was *Microbe Hunters*. We need more books like *Microbe Hunters* to bring a lot more young people into science now.

### Why did you study electrical engineering?

I got As in everything, but I really en-



joyed most the math and physics and chemistry. So why electrical engineering, as opposed to going into physics? Around my family, no one had ever heard of a thing called a physicist. In this middle-class to lower-middle-class culture people were focused on getting a job that would make money, and engineers could get jobs and make money.

I happened to see an advertisement for scholarships being offered by an engineering school in Pittsburgh called Carnegie Institute of Technology. I got a scholarship, so that's what I did. Life is an interesting set of choices, and the decision to go to Carnegie Tech (now Carnegie-Mellon University) was a fantastically good decision.

#### **Something else there got you excited.**

I had a nagging feeling that there was something missing in my courses. There's got to be more to a university education! In the catalog I found a really interesting listing called "Ideas and Social Change," taught by a young new instructor, James March. The first thing he did was to expose us to Von Neumann's and Morgenstern's "Theory of Games and Economic Behavior." Wow! This is mind-blowing! My first published paper was with March in social psychology, on decision-making in small groups.

March introduced me to a more senior and famous professor, Herbert Simon. That led to my taking a course from Simon called "Mathematical Models in the Social Sciences." I got to know Herb, and got to realize that this was a totally extraordinary person.

In January 1956 Herb walked into our seminar of six people and said these famous words: "Over Christmas Allen Newell and I invented a thinking machine." Well, that just blew our minds. He and Newell had formulated the Logic Theorist on December 15th, 1955. They put together a paper program that got implemented in the language called IPL-1, which was not a language that ran on any computer. It was the first list processing language, but it ran in their heads.

#### **That led to your first exposure to computers.**

When we asked Herb in that class, "What do you mean by a machine?"

he handed us an IBM 701 manual, an early IBM vacuum tube computer. That was a born-again experience! Taking that manual home, reading it all night long—by the dawn, I was hooked on computers. I knew what I was going to do: stay with Simon and do more of this. But Carnegie Tech did not have any computers at that time, so I got a job at IBM for the summer of 1956 in New York.

#### **What did you learn at IBM?**

First, plug board programming, which was a phenomenally interesting thing for a geeky kid. Second, the IBM 650, because by that time it became known that Carnegie Tech would be getting a 650. Third, the IBM 704, which was a successor machine to the 701.

When I got back to Carnegie Tech in September 1956 and began my graduate work, there was Alan Perlis, a wonderful computer genius, and later the first Turing Award winner. Perlis was finishing up an amazing program called a compiler. That was "IT," Internal Translator, and it occupied 1,998 words of the 2,000-word IBM 650 drum.

I had known about the idea of algebraic languages because in the summer at IBM someone had come down from the fourth floor to talk to the graduate students and tell them about a new thing that had just hit the scene. You didn't have to write "CLA" for "clear and add," and you didn't have to write "005" for "add." You could write a formula, and a program would translate that formula into machine language. FOR-TRAN. The guy was John Backus,

**This idea has been very important for my career—the experimental approach to computer science as opposed to the theoretical approach.**

who had come downstairs to talk to us. IT's introduction actually preceded Fortran's by about nine months.

#### **What was it like to use a computer then?**

There was no staff between you and the computer. You could book time on the computer, then you went and did your thing. A personal computer! I loved it. I loved the lights, I loved pressing the switches. This idea has been very important for my career—the hands on, experimental approach to computer science as opposed to the theoretical approach. Experiment turns out to be absolutely vital.

I was able to write a rather complicated—for that time—simulation of two companies engaged in a duopolistic decision-making duel about pricing of tin cans in the can industry, the second such simulation of economic behavior ever written. It led to my first conference paper, in December 1958, at the American Economics Association annual meeting.

#### **What did you do for your dissertation?**

A model called EPAM, Elementary Perceiver and Memorizer, a computer simulation model of human learning and memory of nonsense syllables.

I invented a data structure called a Discrimination Net—a memory structure that started out as nothing when the learner starts. List structures had just been invented, but no one had tried to grow trees. I had to, because I would start with two nonsense syllables in the Net, and then the next pair would come in and they'd have to "grow into" the net somewhere. These were the first adaptively growing trees. Now here's an amazing and kind of stupid thing that shows what it means to focus your attention on  $x$  rather than  $y$ . We were focused on psychology. We were not focused on what is now called computer science. So we never published anything about those adaptively growing trees, except as they related to the psychological model. But other people did see trees as a thing to write papers about in the IT literature. So I missed that one!

#### **Where was your first academic job?**

I had wanted to come to the West Coast, and the University of California

at Berkeley was excited about getting me. There I taught two things: organization theory à la March and Simon, and the new discipline called Artificial Intelligence.

There were no books on the subject of AI, but there were some excellent papers that Julian Feldman and I photocopied. We decided that we needed to do an edited collection, so we took the papers we had collected, plus a few more that we asked people to write, and put together an anthology called *Computers and Thought* that was published in 1963.

The two sections mirrored two groups of researchers. There were people who were behaving like psychologists and thinking of their work as computer models of cognitive processes, using simulation as a technique. And there were other people who were interested in the problem of making smart machines, whether or not the processes were like what people were doing.

#### How did choosing one of those lead you to Stanford?

The choice was: do I want to be a psychologist for the rest of my life, or do I want to be a computer scientist? I looked inside myself, and I knew that I was a techno-geek. I loved computers, I loved gadgets, and I loved programming. The dominant thread for me was not going to be what *humans* do, it was going to be what can I make *computers* do.

I had tenure at Berkeley, but the business school faculty couldn't figure out what to make of a guy who is publishing papers in computer journals, artificial intelligence, and psychology. That was the push away from Berkeley. The pull to Stanford was John McCarthy.

#### How did you decide on your research program?

Looking back in time, for reasons that are not totally clear to me, I really, really wanted smart machines. Or I should put the "really" in another place: I really wanted really smart machines.

I wasn't going to get there by walking down the EPAM road, which models verbal learning, or working on puzzle-solving deductive tasks. I wanted to model the thinking processes of scientists. I was interested in problems of induction. Not problems of puzzle

## AI is not much of a theoretical discipline. It needs to work in specific task environments.

solving or theorem proving, but inductive hypothesis formation and theory formation.

I had written some paragraphs at the end of the introduction to *Computers and Thought* about induction and why I thought that was the way forward into the future. That's a good strategic plan, but it wasn't a tactical plan. I needed a "task environment"—a sandbox in which to specifically work out ideas in detail.

I think it's very important to emphasize, to this generation and every generation of AI researchers, how important experimental AI is. AI is not much of a theoretical discipline. It needs to work in specific task environments. I'm much better at discovering than inventing. If you're in an experimental environment, you put yourself in the situation where you can discover things about AI, and you don't have to create them.

#### Talk about DENDRAL.

One of the people at Stanford interested in computer-based models of mind was Joshua Lederberg, the 1958 Nobel Prize winner in genetics. When I told him I wanted an induction "sandbox", he said, "I have just the one for you." His lab was doing mass spectrometry of amino acids. The question was: how do you go from looking at a spectrum of an amino acid to the chemical structure of the amino acid? That's how we started the DENDRAL Project: I was good at heuristic search methods, and he had an algorithm which was good at generating the chemical problem space.

We did not have a grandiose vision. We worked bottom up. Our chemist was Carl Djerassi, inventor of the chemical behind the birth control

pill, and also one of the world's most respected mass spectrometrists. Carl and his postdocs were world-class experts in mass spectrometry. We began to add in their knowledge, inventing knowledge engineering as we were going along. These experiments amounted to titrating into DENDRAL more and more knowledge. The more you did that, the smarter the program became. We had very good results.

The generalization was: in the knowledge lies the power. That was the big idea. In my career that is the huge, "Ah ha!" and it wasn't the way AI was being done previously. Sounds simple, but it's probably AI's most powerful generalization.

Meta-DENDRAL was the culmination of my dream of the early to mid-1960s having to do with theory formation. The conception was that you had a problem solver like DENDRAL that took some inputs and produced an output. In doing so, it used layers of knowledge to steer and prune the search. That knowledge got in there because we interviewed people. But how did the people get the knowledge? By looking at thousands of spectra. So we wanted a program that would look at thousands of spectra and infer the knowledge of mass spectrometry that DENDRAL could use to solve individual hypothesis formation problems.

We did it. We were even able to publish new knowledge of mass spectrometry in the *Journal of the American Chemical Society*, giving credit only in a footnote that a program, Meta-DENDRAL, actually did it. We were able to do something that had been a dream: to have a computer program come up with a new and publishable piece of science.

#### What then?

We needed to play in other playpens. I believe that AI is mostly a qualitative science, not a quantitative science. You are looking for places where heuristics and inexact knowledge can come into play. The term I coined for my lab was "Heuristic Programming Project" because heuristic programming is what we did.

For example, MYCIN was the Ph.D. thesis project of Ted Shortliffe, which turned out to be a very powerful knowledge-based system for diagnosing blood infections and recommending their antibiotic therapies. Lab mem-

bers extracted from Mycin the core of it and called it E-Mycin for Essential Mycin, or Empty Mycin. That rule-based software shell was widely distributed.

What is the meaning of all those experiments that we did from 1965 to 1968? The Knowledge-Is-Power Hypothesis, later called the Knowledge Principle, which was tested with dozens of projects. We came to the conclusion that for the “reasoning engine” of a problem solving program, we didn’t need much more than what Aristotle knew. You didn’t need a big logic machine. You need *modus ponens*, backward and forward chaining, and not much else in the way of inference. *Knowing a lot* is what counts. So we changed the name of our laboratory to the “Knowledge System Lab,” where we did experiments in many fields.

#### What other AI models did you use?

AI people use a variety of underlying problem-solving frameworks, and combine a lot of knowledge about the domain with one of these frameworks. These can either be forward-chaining—sometimes called generate and test—or they could be backward-chaining, which say, for example, “here’s the theorem I want to prove, and here’s how I have to break it down into pieces in order to prove it.”

I began classified research on detecting quiet submarines in the ocean by their sound spectrum. The problem was that the enemy submarines were very quiet, and the ocean is a very noisy place. I tried the same hypothesis formation framework that had worked for DENDRAL, and it didn’t even come close to working on this problem.

Fortunately Carnegie Mellon people—Reddy, Erman, Lesser and Hayes-Roth—had invented another framework they were using for understanding speech, the Blackboard Framework. It did not work well for them, but I picked it up and adapted it for our project. It worked beautifully. It used a great deal of knowledge at different “levels of abstraction.” It allowed flexible combination of top-down and bottom-up reasoning from data to be merged at those different levels. In Defense Department tests, the program did better than people.

But that research was classified as “secret.” How could ideas be pub-

## In my view the science that we call AI, maybe better called computational intelligence, is the manifest destiny of computer science.

lished from a military classified project? The Navy didn’t care about the blackboard framework; that was computer science. So we published the ideas in a paper on a kind of hypothetical: “how to find a koala in eucalyptus trees,” which was a non-classified problem drawn from my personal experience in an Australian forest!

#### Talk about being an entrepreneur as well as an academic.

There was a very large demand for the software generalization of the MYCIN medical diagnosis expert system “shell,” called EMYCIN. So a software company was born called Teknowledge, whose goal was to migrate EMYCIN into the commercial domain, make it industrial strength, sell it, and apply it. Teknowledge is still in existence.

Our Stanford MOLGEN project was the first project in which computer science methods were applied to what is now called computational molecular biology. Some MOLGEN software turned out to have a very broad applicability and so was the basis of the very first company in computational molecular biology, called Intelligenetics, later Intellicorp. They had lots of very sophisticated applications. During the dot-com bust they went bust, but they lasted, roughly speaking, 20 years.

#### In the 1980s you studied the Japanese government’s major effort in AI.

The Japanese plan was very ambitious. They organized a project to essentially do knowledge-based AI, but in a style different from the style we were accustomed to in this country. For one thing,

they wanted to do it in the “I-am-not-LISP style,” because the Japanese had been faulted in the past for being imitators. So they chose Prolog and tried formal methods. And they included parallel computing in their initiative.

They made a big mistake in their project of not paying enough attention to the application space at the beginning. They didn’t really know what applications they were aiming at until halfway through; they were flying blind for five years. Then they tried to catch up and do it all in five more years, and didn’t succeed. [See the book, *The Fifth Generation*,” written with Pamela McCorduck].

#### How did you come to work for the U.S. government?

In 1994 an amazing thing happened. The phone rings and it is Professor Sheila Widnall of the Department of Aeronautics and Astronautics of MIT. She said, “Do you know anyone who wants to be Chief Scientist of the Air Force? And by the way, if you are interested let me know.” She had been chosen to be Secretary of the Air Force, and she was looking for her Chief Scientist. I thought about it briefly, told her yes, and stayed for three years.

My job was to be a window on science for the Chief of Staff of the Air Force. I was the first person to be asked to be Chief Scientist who was not an Aero-Astro person, a weapons person, or from the physical sciences. There had not been any computer scientists before me.

I did two big things. One was consciousness-raising in the Air Force about software. The one big report I wrote, at the end of my term, was a report called, *It’s a Software-First World*. The Air Force had not realized that. They probably still do not think that. They think it is an airframe-based world.

The other was on software development. The military up to that point believed in, and could only imagine, a structured-programming top-down world. You set up requirements, you get a contractor to break down the requirements into blocks, another contractor breaks them down into mini-blocks, and down at the bottom there are some people writing the code. It takes years to do. When it all comes back up to the top, (a) it’s not right,



and (b) it's not what you want anymore. They just didn't know how to contract for cyclical development. Well, I think we were able to help them figure out how to do that.

### **What happened after your "tour of duty" in Washington?**

It was a rather unsettling experience to come back to Stanford. After playing a role on a big stage, all of a sudden you come back and your colleagues ask, "What are you going to teach next year? Intro to AI?"

So at the beginning of 2000, I retired. Since then I have been leading a wonderful life doing whatever I please. Now that I have a lot more time than I had before, I'm getting geekier and geekier. It feels like I'm 10 years old again, getting back involved with details of computing.

The great thing about being retired is not that you work less hard, but that what you do is inner-directed. The world has so many things you want to know before you're out of here that you have a lot to do.

### **Why is history important?**

When I was younger, I was too busy for history and not cognizant of the importance of it. As I got older and began to see my own career unfolding, I began to realize the impact of the ideas of others on my ideas. I became more and more of a history buff.

That convinced me to get very serious about archives, including my own. If you're interested in discoveries and the history of ideas, and how to manufacture ideas by computer, you've got to treat this historical material as fundamental data. How did people think? What alternatives were being considered? Why was the movement from one idea to another preposterous at one time and then accepted?

### **You are a big fan of using heuristics not only for AI, but also for life. What are some of your life heuristics?**

► Pay a lot of attention to empirical data, because in empirical data one can discover regularities about the world.

► Meet a wonderful collaborator—for me it was Joshua Lederberg—and work with that collaborator on meaningful problems

► It takes a while to become really,

really good at something. Stick with it. Focus. Persistence, not just on problems but on a whole research track, is really worth it. Switching in the middle, flitting around from problem to problem, isn't such a great idea.

► Life includes of a lot of stuff you have to do that isn't all that much fun, but you just have to do it.

► You have to have a global vision of where you're going and what you're doing, so that life doesn't appear to be just Brownian motion where you are being bumped around from one little thing to another thing.

### **How far have we come in your quest to have computers think inductively?**

Our group, the Heuristic Programming Project, did path-breaking work in the large, unexplored wilderness of all the great scientific theories we could possibly have. But most of that beautiful wilderness today remains largely unexplored. Am I am happy with where we have gotten in induction research? Absolutely not, although I am proud of the few key steps we took that people will remember.

### **Is general pattern recognition the answer?**

I don't believe there is a general pattern recognition problem. I believe that pattern recognition, like most of human reasoning, is domain specific. Cognitive acts are surrounded by knowledge of the domain, and that includes acts of inductive behavior. So I don't really put much hope in "general anything" for AI. In that sense I have been very much aligned with Marvin Minsky's view of a "society of mind." I'm very much oriented toward a knowledge-based model of mind.

### **How should we give computers knowledge?**

I think the only way is the way human culture has gotten there. We transmit our knowledge via cultural artifacts called texts. It used to be manuscripts, then it was printed text, now it's electronic text. We put our young people through a lot of reading to absorb the knowledge of our culture. You don't go out and experience chemistry, you study chemistry.

We need to have a way for computers to read books on chemistry and learn

chemistry. Or read books on physics and learn physics. Or biology. Or whatever. We just don't do that today. Our AI programs are handcrafted and knowledge engineered. We will be forever doing that unless we can find out how to build programs that read text, understand text, and learn from text.

Reading from text in general is a hard problem, because it involves all of common sense knowledge. But reading from text in structured domains I don't think is as hard. It is a critical problem that needs to be solved.

### **Why is AI important?**

There are certain major mysteries that are magnificent open questions of the greatest import. Some of the things computer scientists study are not. If you're studying the structure of databases—well, sorry to say, that's not one of the big magnificent questions.

I'm talking about mysteries like the initiation and development of life. Equally mysterious is the emergence of intelligence. Stephen Hawking once asked, "Why does the universe even bother to exist?" You can ask the same question about intelligence. Why does intelligence even bother to exist?

We should keep our "eye on the prize." Actually, two related prizes. One is that when we finish our job, whether it is 100 years from now or 200 years from now, we will have invented the ultra-intelligent computer. The other is that we will have a very complete model of how the human mind works. I don't mean the human brain, I mean the mind: the symbolic processing system.

In my view the science that we call AI, maybe better called computational intelligence, is the manifest destiny of computer science.

For the people who will be out there years from now, the question will be: will we have fully explicated the theory of thinking in your lifetime? It would be very interesting to see what you people of a hundred years from now know about all of this.

**It will indeed. Stay tuned.** 

Len Shustek (shustek@computerhistory.org) is the chairman of the Computer History Museum.

Copyright held by author.

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

**Elastic computing has great potential,  
but many security challenges remain.**

BY DUSTIN OWENS

# Securing Elasticity in the Cloud

AS SOMEWHAT OF a technology-hype curmudgeon, I was until very recently in the camp that believed cloud computing was not much more than the latest marketing-driven hysteria for an idea that has been around for years. Outsourced IT infrastructure services, aka Infrastructure as a Service (IaaS), has been around since at least the 1980s, delivered by the telecommunication companies and major IT outsourcers. Hosted applications, aka Platform as a Service (PaaS) and Software as a Service (SaaS), were in vogue in the 1990s in the form of application service providers (ASPs).

Looking at cloud computing through this perspective had me predicting how many more months it would be before the industry came up with another “exciting” technology with which to generate mass confusion and buzz. However, I have recently been enlightened as to the true potential of cloud computing and have become very excited

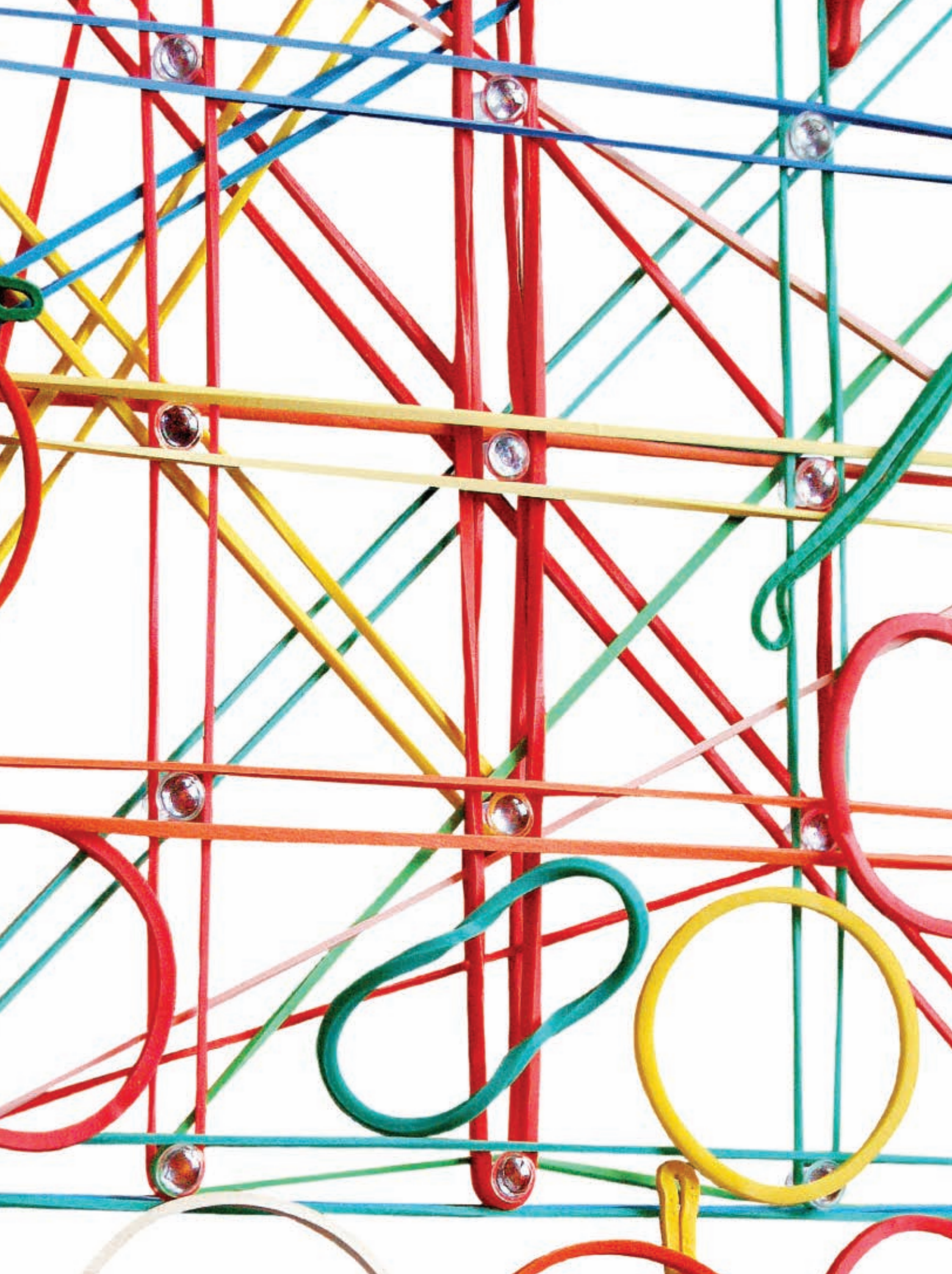
about it, to say the least. This concept, which has generated the most industry hype in years—and which has executives clamoring for availability because of promises of substantial IT cost savings and innovation possibilities—has finally won me over.

So, what did I discover about cloud computing that has made a convert out of someone who was so adamantly convinced that it was nothing more than the latest industry topic du jour? First let me explain that it was no small feat. It took a lot of work to sort through the amazing amount of confusion concerning the definition of cloud computing, let alone find a nugget of real potential. Definitions abound, and with my curmudgeon hat still solidly in place I was beginning to see a lot of hair-splitting and “me too” definitions that just seemed to exacerbate the problem. I finally settled on the definition provided by the National Institute of Standards and Technology (NIST) because of the simplicity the framework provides (see the accompanying sidebar). Still, it wasn’t until a good friend who had already discovered the true potential hidden in all this madness provided me with some real-world use cases for *elasticity* that the light began shining very brightly.

Elasticity, in my very humble opinion, is the true golden nugget of cloud computing and what makes the entire concept extraordinarily evolutionary, if not revolutionary. NIST’s definition of elasticity (<http://csrc.nist.gov/groups/SNS/cloud-computing/>) is as follows: “Capabilities can be rapidly and elastically provisioned, in some cases automatically, to quickly scale out and rapidly released to quickly scale in. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be purchased in any quantity at any time.” When elasticity is combined with on-demand self-service capabilities it could truly become a game-changing force for IT.

Advanced outsourced IT infrastructure and software services, once available only to organizations with large









budgets available to develop, build, and support ongoing use of these resources, can now be provided to small to medium organizations. In addition, these resources can be added, changed, or removed much more rapidly, potentially allowing for exponential advances in operational efficiency. These sorts of changes to major IT services environments that previously (and for the most part currently) took months if not years to plan and execute might be done in a matter of minutes or hours if elasticity holds up to its promise. In other words, elasticity could bring to the IT infrastructure what Henry Ford brought to the automotive industry with assembly lines and mass production: affordability and substantial improvements on time to market.

Enlightening as this realization has been, it has also become clear that several monumental security challenges (not to mention many monumental nonsecurity-related challenges, not least of which are full functionality availability and how well an organization's environment is prepared to operate in a distributed model) now come into play and will need to be addressed in order for the elasticity element of cloud computing to reach its full potential. Most of the dialogue I am engaged in with customers today and that I see in publicized form, however, is simplistically centered on security challenges with IT outsourcing in general. These are challenges have existed for some time in the predecessor models mentioned earlier: who within an outsourcer is able to access a customer's data, perimeter security considerations when outsourcing, DOS/DDOS (denial of service/distributed denial of service), resource starvation, and compliance challenges with where data is stored or backed up. These are all challenges that I have provided counsel on for many years and are nothing new or insurmountable. Don't misunderstand me. These challenges are indeed very real and still need to be addressed, but I strongly believe most should be fairly well known by now and can be readily met through existing procedural or technological solutions.

The challenges I am more concerned about are those introduced by adding elasticity and on-demand self-



**Elasticity, in my very humble opinion, is the true golden nugget of cloud computing and what makes the entire concept extraordinarily evolutionary, if not revolutionary. Elasticity could bring to the IT infrastructure what Henry Ford brought to the automotive industry with assembly lines and mass production: affordability and substantial improvements on time to market.**



service to form the full extent of cloud computing—those elements that in my opinion make a particular service something more than a just an outsourced service with a prettier marketing face.

### **Elasticity Security Challenges**

Enabling elasticity in the cloud strongly implies the use of virtualization. Though the inherent security challenges in virtualization are certainly not new, how it is likely to be used by cloud-computing providers to achieve elastic IT environments on a grand scale poses some interesting security challenges worth exploring in more detail. In addition, as virtualization technology continues to evolve and gain popularity, so does the discovery of new vulnerabilities; witness the recently announced vulnerability (<http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2009-3733>) whereby one is able to traverse from one virtual machine (VM) client environment to other client environments being managed by the same hypervisor.

These new vulnerabilities could have significantly greater impacts in the cloud-computing arena than within an organization's corporate environment, especially if not dealt with expeditiously. Case in point: imagine that many customers are being managed by a single hypervisor within a cloud provider. The vulnerability shared above might allow a customer to access the virtual instances of other customers' applications if not addressed. Consider the impact if your bank or particularly sensitive federal government or national defense information happen to be managed in this sort of environment, and the cloud provider does not immediately deal with, or even know about, a vulnerability of this nature.

With this bit of background, it is clear that providing adequate administrative separation between virtual customer environments will be a significant security challenge with elasticity. Cloud providers will need to be prepared to account for and show how their particular services are able to control vulnerabilities such as the earlier example and keep similar yet-to-be discovered vulnerabilities from having devastating impacts on their custom-

ers. Perhaps more importantly, *critical infrastructure* (see [http://en.wikipedia.org/wiki/Critical\\_infrastructure](http://en.wikipedia.org/wiki/Critical_infrastructure) for definition) could be subject to insurmountable risk and/or loss of sensitive information if providers lack the necessary controls. As services offered from the cloud continue to mature and expand, the threat posed is not limited to unauthorized information access but may include any cloud-provided computing systems (such as virtual servers, virtual desktops, and so on). We hope the U.S. government recognizes and addresses this challenge as federal agencies move rapidly toward adoption of cloud-based services (<http://www.federalnewsradio.com/?sid=1836091&nid=35>), because the potential consequences are particularly unsettling.

Addressing this challenge may be no small feat. For one, in order for cloud providers to minimize their management costs and obtain profitability, they are expected to have to use shared administrative management systems (that is, hypervisors) across multiple virtual customer environments. I can envision certain service models where this theory may not hold true: for example, if each customer were given sole hypervisor (or hypervisor-like) management access that connected only to that customer's virtual environment, such as within a virtual private cloud offering. Use of a separate management system for every customer in every service model is probably not realistic simply because of cost containment.

In researching several cloud providers' capabilities in this regard, I could not clearly see how their solutions could effectively address the entirety of the provided traversal vulnerability example when multiple customers are using the same hypervisor, at least at the time of writing this article. Although some provide detail of built-in software functionality within their hypervisors meant to curtail one customer from gaining access to another's environment, I suspect these capabilities would not fully address the vulnerability in question and are certainly worthy of further detailed review.

Another interesting challenge with elasticity in the cloud will be in the ability to provide fine-grained access and predefined security controls

across the entirety of a virtual customer environment. The service models to which this might apply most directly are those that provide IaaS and PaaS functionality such as dynamic multilevel security services or multitier application environments. To understand the challenge better, it is probably useful to provide some context for how these types of services are built and administered in today's corporate infrastructure, such as with a multitier application. One example of a typical scenario is where the application development group needs to work closely with the network and hopefully IT security groups to establish proper communication paths among the various tiers, including limiting which network protocols are allowed to interface with each of the tiers. This would be done to ensure proper routing of information and to limit the attack surface available to hackers or malware once the system is put into production.

In addition, when dealing with certain types of data such as financial or credit cards, certain regulations and industry standards have a requirement for separation of duties to aid in protection from certain scenarios—for example, an application developer inserting code into software that would allow skimming of financial data and not having an audit trail available as the developer elected not to enable one for obvious reasons. Although various cloud providers do provide some detail on how their solutions address this concern, proper implementation by the user organization, as well as performing due diligence review of actual capabilities within a desired delivery model, will be critical to ensuring this challenge can be adequately addressed.

Fast forward to the cloud scenario in which a developer now has access to a self-service portal where in a few mouse clicks he or she would be able to build out a new multitier virtual application environment. Without fine-grained access controls available through the self-service portal it will be extremely difficult to enforce separation of duties to keep this developer from accessing sensitive data he or she shouldn't have access to, or promoting new code to production without having gone through proper security review or

change management. In this scenario, the application could be extremely vulnerable to attack or even inadvertently cause a production application to cease operating properly. The ability to implement and enforce access controls to a granular level, defining who has the authority to perform which actions within these environments, will be absolutely necessary.

Having the ability to predefine security control templates may also aid in this sort of scenario. This means the organization's IT security group is able to define a set of controls that must be applied to a given application depending on the type of data it will be processing or how the application will be used. For example, as the developer builds out the new virtual environment that processes credit-card information, the self-service portal might identify the type of data to be processed and apply predefined security controls to the database, application, and Web front end, as well as predefined firewall rule sets limiting network access to the various tiers. It is unlikely that this capability exists today, anywhere, and we are probably years away from ubiquitous availability.

Another security challenge that develops out of this scenario and in the same vein is how to enforce proper configuration and change management in this more dynamic and elastic model. Even where a portal is capable of granular-access controls that control which actions a given user is able to perform, it also needs to enforce *when* and *under what circumstances* a user is allowed to perform certain actions. Without this ability, untested code or system changes could result in business-impacting (or even devastating) results. Even something as "slight" as rolling a new system into production without ensuring that proper server and application patches have been applied could result in significant damage to an organization. Therefore, a mechanism within self-service portals for enforcing an organization's change policies becomes a worthy and necessary capability.

These are but a few of the challenges that come to mind within a truly elastic PaaS and/or IaaS service model and not even delving into separate challenges with SaaS. Other chal-

# The NIST Definition of Cloud Computing

By Peter Mell and Tim Grance

Cloud computing is still an evolving paradigm. Its definitions, use cases, underlying technologies, issues, risks, and benefits will be refined in a spirited debate by the public and private sectors. These definitions, attributes, and characteristics will evolve and change over time. The cloud-computing industry represents a large ecosystem of many models, vendors, and market niches. The following definition attempts to encompass all of the various cloud approaches.

Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (for example, networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service-provider interaction. This cloud model promotes availability and is composed of five essential characteristics, three service models, and four deployment models.

## ESSENTIAL CHARACTERISTICS

**On-demand self-service.** A consumer can unilaterally provision computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with each service's provider.

**Broad network access.** Capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms (for example, mobile phones, laptops, and PDAs).

**Resource pooling.** The provider's computing resources are pooled to serve multiple consumers using a multitenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand. There is a sense of location independence in that the customer generally has no control or knowledge over the exact location of the provided resources but may be able to specify location at a higher level of abstraction (for example, country, state, or data

center). Examples of resources include storage, processing, memory, network bandwidth, and virtual machines.

**Rapid elasticity.** Capabilities can be rapidly and elastically provisioned, in some cases automatically, to quickly scale out and rapidly released to quickly scale in. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be purchased in any quantity at any time.

**Measured service.** Cloud systems automatically control and optimize resource use by leveraging a metering capability at some level of abstraction appropriate to the type of service (for example, storage, processing, bandwidth, and active user accounts). Resource usage can be monitored, controlled, and reported, providing transparency for both the provider and consumer of the utilized service.

## SERVICE MODELS

**Cloud SaaS (Software as a Service).** The capability provided to the consumer is to use the provider's applications running on a cloud infrastructure. The applications are accessible from various client devices through a thin client interface such as a Web browser (for example, Web-based email). The consumer does not manage or control the underlying cloud infrastructure, including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited user-specific application configuration settings.

**Cloud PaaS (Platform as a Service).** The capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages and tools supported by the provider. The consumer does not manage or control the underlying cloud infrastructure, including network, servers, operating systems, or storage, but has control over the deployed applications and possibly application-hosting environment configurations.

**Cloud IaaS (Infrastructure as a Service).**

The capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, deployed applications, and possibly limited control of select networking components (for example, host firewalls).

## DEPLOYMENT MODELS

**Private cloud.** The cloud infrastructure is operated solely for an organization. It may be managed by the organization or a third party and may exist on or off premise.

**Community cloud.** The cloud infrastructure is shared by several organizations and supports a specific community that has shared concerns (for example, mission, security requirements, policy, and compliance considerations). It may be managed by the organizations or a third party and may exist on or off premise.

**Public cloud.** The cloud infrastructure is made available to the general public or a large industry group and is owned by an organization selling cloud services.

**Hybrid cloud.** The cloud infrastructure is a composition of two or more clouds (private, community, or public) that remain unique entities but are bound together by standardized or proprietary technology that enables data and application portability (for example, cloud bursting for load balancing between clouds).

**Note:** Cloud software takes full advantage of the cloud paradigm by being service oriented with a focus on statelessness, low coupling, modularity, and semantic interoperability.

Peter Mell and Tim Grance are with the National Institute of Standards and Technology, Information Technology Laboratory, Gaithersburg, MD.

lenges include the ability to provide audit trails across these environments for regulatory compliance and digital forensic purposes, enforcement, and awareness of differing levels of zones among development, test, and production environments to protect the integrity of services deployed in the higher-level environments, as well as controlling whom is authorized to ex-

pand or contract a service within one of these environments. This last challenge could pose particular financial issues in the elastic "pay by the drink" service model if, for example, users are able to add services at will and an organization gets a bill at the end of the month for excessive service additions.

Changing tack slightly, however, it is worth mentioning the challenges in

providing adequate levels of security services within nonsecurity-related environments. One of these challenges is with traditionally nonsecurity-minded providers needing to supply service options for common security capabilities such as intrusion detection, firewalls, content filtering, and vulnerability testing. In predecessor service models, such as an ASP, these



services could be offered through partnerships with security vendors and manually designed and provisioned into the outsourced environment. In the new model, however, how providers are able to provide tighter integration with these services in order not to lose full elasticity may be interesting. It may require creating optional service hooks from a provider's self-service portal to security service products or perhaps developing interesting but complex multiservice cloud models provided by multiple specialty service providers. Either way, this challenge is probably worthy of a discussion in and of itself because of the perceived number of additional issues it brings to mind. Note that some vendors do offer these capabilities today, particularly within virtual private cloud models, but of the vendors researched, none is fully addressing for every model it offers.

Encryption capabilities for data-at-rest may be an interesting challenge as well. For example, given the previous environment traversal example, use of file-based encryption within a virtual environment would be essentially worthless in offering protection from remote access. If one can readily gain access to another's environment, this would also provide access to any front-end encryption mechanism used for file-based encryption within the virtual environment. Disk-based encryption becomes particularly challenging because of the nature of virtual storage and potential lack of user organizational control over where data may be physically stored (which disk does one encrypt for a given customer and other constraints in sharing of physical disks among multiple customers). It will certainly be necessary to explore a prospective provider's capabilities for encrypting data-at-rest and how well it addresses the shared concerns, especially for those organizations with regulatory requirements dictating the use of file- and/or disk-based encryption.

It should be apparent by now that cloud computing is fraught with a number of security challenges. While some of the concepts and scenarios discussed here are focused on more advanced service models, the intent is to create a bit more awareness of

**Though the inherent security challenges in virtualization are not new, how it is likely to be used by cloud-computing providers to achieve elastic IT environments on a grand scale poses some interesting security challenges.**

what the industry will be faced with in moving toward these new models that offer greater levels of "true" cloud computing. Depending on the type of service model being discussed and various use cases, exploring all of the challenges is all but impossible, especially not in a single discussion. In addition, some of the security challenges discussed appear to be recognized by certain cloud providers but are primarily being addressed through the use of private cloud models (Amazon and OpSource are two such vendors offering answers within a virtual private cloud offering), suggesting perhaps higher costs versus a public cloud offering and/or limited availability in addressing within other cloud-delivery models.

The promise of what an elastic cloud-computing model could do for the IT world, however, is extremely invigorating and certainly worth pursuing. It can only be hoped that organizations already taking this path or seriously considering doing so will take the time to fully appreciate the security challenges facing them and whether or not adoption at this point fits into their risk appetite. Certainly, keeping these and other security challenges in mind while assessing how a prospective cloud provider can address these concerns (and at what cost and with what deployment constraints) should be a critical business objective. ■

#### **Q** Related articles on [queue.acm.org](http://queue.acm.org)

##### **Cybercrime 2.0: When the Cloud Turns Dark**

*Niels Provos, Maheeb Abu Rajab,  
Panayiotis Mavrommatis*

<http://queue.acm.org/detail.cfm?id=1517412>

##### **Meet the Virts**

*Tom Killalea*

<http://queue.acm.org/detail.cfm?id=1348589>

##### **CTO Roundtable: Cloud Computing**

*Mache Creeger*

<http://queue.acm.org/detail.cfm?id=1536633>

**Dustin Owens** ([dustin.owens@bt.com](mailto:dustin.owens@bt.com)) is a senior principal consultant with BT Americas' Business Innovation Group. He provides consulting services centered on operational risk and security management for multinational customers, specializing in applying these concepts to various areas of strategic business innovation. He has more than 14 years of practical experience in addressing information security within distributed computing environments.

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

**Emulating a video system shows how even a simple interface can be more complex—and capable—than it appears.**

BY GEORGE PHILLIPS

# Simplicity Betrayed

AN EMULATOR IS a program that runs programs built for different computer architectures from the host platform that supports the emulator. Approaches differ, but most emulators simulate the original hardware in some way. At a minimum the emulator interprets the original CPU instructions and provides simulated hardware-level devices for input and output. For example, keyboard input is taken from the host platform and translated into the original hardware format, resulting in the emulated program “seeing” the same sequence of keystrokes. Conversely, the emulator will translate the original hardware screen format into an equivalent form on the host machine.

The emulator is similar to a program that implements the Java Virtual Machine (JVM). The difference is merely one of degree. JVM is designed to enable efficient and tractable implementations, whereas an emulator’s machine is defined by real hardware that generally imposes undesirable constraints on the emulator. Most significantly, the original hardware may be fully described only in terms

of how existing software uses it. JVM tends to be forward-looking with the expectation that new code will be written and run under JVM to increase its portability. Emulators tend to be backward-looking, expecting only to make old code more portable.

In either case the emulator and JVM give the programs that run under them a suite of services needed to interact with the host system. JVM presents those services as a series of API calls; the emulator presents them as simulated hardware. Nonetheless, the simulated hardware is an API—just not in the form most programmers expect.

## TRS-80 Example

As an example hardware API, consider the TRS-80 video system, which displays text and graphics on a modified television set. It has 16 lines of characters with 64 columns each. It supports the normal ASCII character set, with an additional 64 graphics characters allowing every possible combination of a 2-pixel by 3-pixel sub-block. Judicious use of the graphics characters provide an effective 128-pixel by 48-pixel resolution, albeit with pixels the size of watermelon seeds. A TRS-80 program displays a character by writing the character value to the memory location associated with the desired position. In effect, the API has only one call:

```
void ChangeCharacter
(location /* 0 - 1024 */,
character)
```

Emulating such a simple graphics format is trivial. It can be rendered quite adequately on a 512-by-192 image allotting each character an 8-by-12 rectangle. Each graphics pixel is a 4-by-4 rectangle, while the characters themselves occupy the upper two-thirds, or an 8-by-8 area. While you could get away with any old font for the characters, a little more work will get something that looks dot-for-dot identical to the original. (To get the







same aspect ratio as the original, the image should be doubled in height. We'll keep the numbers as is to simplify exposition.)

Figure 1 shows how an emulator converts the hardware-level character values into an actual image representation. While the figure serves as a blueprint for the emulator, it also shows what a TRS-80 program must do in order to display something. One detail is missing: changes to screen memory are not displayed instantaneously. The hardware redraws the display every  $1/60^{\text{th}}$  of a second. This means the emulator must assemble the static image as described and dis-

play it every  $1/60^{\text{th}}$  of a second.

The resulting emulation will look very much like the original. The majority of programs that run under the emulator will appear exactly the same as when run on original hardware, but if you pay very close attention you will see some differences. Watch as the screen is cleared from all black to all white. In both the original and the emulation you get a tear, because filling the screen takes long enough that it goes over more than one video frame. Just before the fill starts, the frame will be black. The fill is partially done on the next frame, and the display shows white at the top and black

at the bottom. That happens for just an instant; by the next frame the filling is done and the frame is filled with white.

Even though this is at the edge of perception, the tear exhibited by the two is quite different. You will need a video camera to see it in the original, but the emulator can either dump frames for offline analysis or be told to step one frame at a time. Figure 2 shows the difference between the tear on the original hardware and that seen in the emulator.

Although apparently minor, this difference is puzzling. The emulator implemented the specification exactly as written, and no bugs were found in the code. Moreover, the specification was obviously correct and complete. Except for the evidence at hand, the situation is impossible. Of course, the problem lies in the specification, which only appears complete. The assumption that a particular character is either there or not is incorrect. The hardware does not draw a character at a time; it draws one line of a character at a time. That character has the opportunity to change after the drawing has already started. The tearing on the original results from a character being blank on the initial passes and subsequently filled in. Put another way, the character is not atomic but made up of 12 pieces stacked on top of one another; each piece is 8-by-1. Incidentally, those 8-by-1 pieces are atomic—they are displayed entirely or not at all. The graphics hardware ends up reading each displayed character 12 times.

Refining the emulator to correct this difference is straightforward. Instead of waiting an entire  $1/60^{\text{th}}$  of a second before drawing the screen, it will be drawn a line at a time. With 192 lines the emulation loop looks something like this:

```
for (i = 0; i < 192; i++) {
    emulate CPU for 86.8 micro-seconds
    draw line i of the video display
}
```

Now the tearing on the emulator is the same as the hardware. You may be tempted to declare the specification and the emulator complete be-

Figure 1. Translating TRS-80 screen memory into a displayed image.

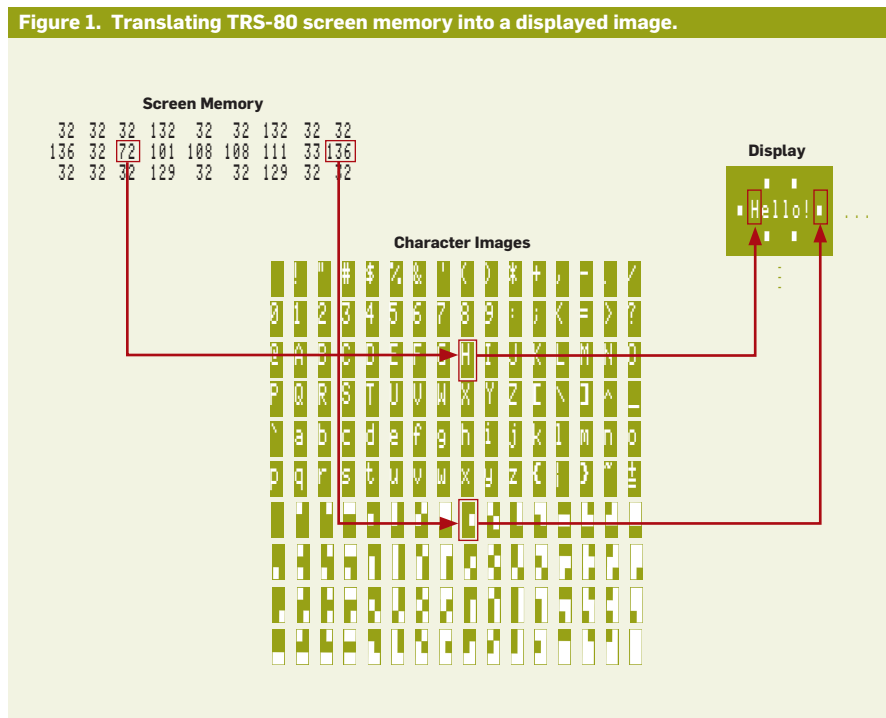


Figure 2. Difference in tears between the emulation and the original hardware.



cause of the major increase in output fidelity. As a conscientious developer, however, your reaction must be exactly the opposite. A rather small test case required a considerable change in the program. Now is the time to investigate further and look for additional bugs. In all likelihood the specification needs more refinement. At the very least, a better test case for the new functionality is needed. After a bit of thought it becomes clear that displaying a one-line-high pixel (1-by-4) would make such a test case.

This can be done in three simple steps.


1. Write an ordinary 4-by-4 pixel on screen.
2. Wait until the first line has been drawn by the graphics hardware.
3. Quickly erase the pixel.

All that will be visible on screen is the 1-by-4 part of the pixel that was drawn before you pulled the rug out from under the 4-by-4 pixel. Many pixels can be combined to create something seemingly impossible on a stock TRS-80: a high-resolution diagonal line.


The only thing missing is some way of knowing which line the hardware is drawing at any one point. Fortunately, the graphics hardware generates an interrupt when it draws a frame. When that interrupt happens you know exactly where the graphics hardware is. A few difficulties of construction remain, but they come down to trivial matters such as putting in delays between the memory accesses to ensure you turn pixels on and off in step with each line being drawn.

Here the emulator is a boon. Making such a carefully timed procedure work on real hardware is very difficult. Any mistake in timing will result in either no display because a pixel was erased too quickly or a blocky line caused by erasing pixels too slowly. Not only does the debugger not care about time, it eschews it entirely. Single-stepping through the code is useless. To be fair, the debugger cannot single-step the graphics hardware. Even if it did, the phosphor would fade from sight before you could see what was happening.

The emulator can single-step the



**The majority of programs that run under the emulator will appear exactly the same as when run on original hardware, but if you pay very close attention you will see some differences.**



processor and the graphics. It can show exactly what is being drawn and point out when screen memory writes happen at the incorrect times. In no time at all a demonstration program is written that shows a blocky line in a simple emulator and a diagonal line in a more accurate emulator (see Figure 3).

### The Real Machine

The program is impressive as it must read/write to the display with micro-second-level timing. The real excitement is running the program on the original machine. After all, the output of the emulator on a PC is theoretically compelling but it is actually producing graphics that pale in comparison to anything else on the platform. On the real machine it will produce something never before seen.

Sadly, the program utterly fails to work on the real machine. Most of the time the display is blank. It occasionally flashes the ordinary block line for a frame, and very rarely one of the small pixels shows up as if by fluke.

Once again, the accurate emulation is not so accurate. The original tearing effect proves that the fundamental approach is valid. What must be wrong is the timing itself. For those strong in software a number of experimental programs can tease out the discrepancies. Hardware types will go straight to the schematic diagrams that document the graphics hardware in detail. Either way, several characteristics will become evident:

- ▶ Each line takes 64 microseconds, not 86.8.
- ▶ There are 264 lines per frame; 192 visible and 72 hidden.
- ▶ A frame is 16,896 microseconds or 59.185 frames per second, not 60.

What's most remarkable is how the emulator appeared to be very accurate in simulating a tear when it was, in fact, quite wrong. So much has been written about the brittleness of computer systems that it is easy to forget how flexible and forgiving they can be at times. The numbers bring some relief to the emulator code itself. What appeared to be floating-point values for timing are in fact just multiples of the system clock. Simple, integer relationships exist between the speed of the CPU and the graphics hardware.

# A Z-80 Cycle Waster

The Z-80 code is in the comments alongside equivalent C code. The C program is self-contained and runs an exhaustive test verifying that waitHL() always uses  $H * 256 + L + 100$  cycles. Observe that the JR conditional branch instructions take extra time if the branch is taken. Those time differences along with looping are used to expand the subroutine's running time in proportion to the requested number of cycles.

```
// Z-80 registers are akin to C global variables.

unsigned char A, H, L;
unsigned char C, Z;

int cycles;

void t(int n)
{
    cycles += n;
}

void waitA();

/* Cycle Cost   C code           Z-80 code */

void wait256()
{
wHL256:
    t(4);   H--; Z = H != 0;      // DEC H
    t(7);   A = 127;             // LD  A,127
    t(17);  waitA();             // CALL wA
}

void waitHL()
{
wHL:
    t(4);   H++; Z = H == 0;     // INC H
    t(4);   H--; Z = H == 0;     // DEC H
    t(7);   if (!Z) {           // JR  NZ,wHL256
    t(5);       wait256();
                goto wHL;
    }
    t(4);   A = L;              // LD  A,L
    t(0);   waitA();
}

void waitA()
{
wA:
    t(4);   C = A & 1;         // RRCA
            A = (A << 7) | (A >> 1);
    t(7);   if (C) {           // JR  C,have1
    t(5);       goto have1;
    }
    t(4);                               // NOP
have1:
    t(4);   C = A & 1;         // RRCA
            A = (A << 7) | (A >> 1);
    t(7);   if (!C) {         // JR  NC,no2
    t(5);       goto no2;
    }
    t(7);   if (!C) {         // JR  NC,no2
    t(5);       goto no2;
    }
}

no2:
    t(4);   C = A & 1;         // RRCA
            A = (A << 7) | (A >> 1);
    t(7);   if (!C) {         // JR  NC,no4
    t(5);       goto no4;
    }
    t(5);   if (!C) {         // RET NC
    t(6);       return;
    }
    t(4);                               // NOP
no4:
    t(4);   C = A & 1;         // RRCA
            A = (A << 7) | (A >> 1);
    t(7);   if (!C) {         // JR  NC,no8
    t(5);       goto no8;
    }
    t(13);  /* *0 = A; */     // LD  (0),A
no8:
    t(7);   A &= 15;          // AND A,15
            Z = A == 0;
    t(5);   if (Z) {         // RET Z
    t(6);       return;
    }
wait16:
    t(4);   A--; Z = A == 0;   // DEC A
    t(7);   if (!Z) {         // JR  NZ,wait16
    t(5);       goto wait16;
    }
    t(5);   if (Z) {         // RET Z
    t(6);       return;
    }
}

//-----

#include <stdio.h>

int main(int argc, char *argv[])
{
    for (int hl = 0; hl < 65536; hl++)
    {
        H = hl / 256;
        L = hl & 255;
        cycles = 0;
        waitHL();

        if (cycles != hl + 100)
        {
            printf("Blew it on %d (got
            %d instead of %d)\n", hl,
            cycles, hl + 100);
        }
    }

    return 0;
}

```



We can restate timings from the CPU's perspective:

- ▶ Each line takes 128 cycles.
- ▶ The hidden 72 lines go for 9,216 cycles.
- ▶ Each frame is 33,792 cycles (264 \* 128).

The number of frames per second is still a floating-point number, but the emulator core can return to integers as you might expect for a digital system.

With the new timings in place, the emulator exhibits the same problems as the real hardware. With a bit of (tedious) fiddling with the timing, the program almost works on the real hardware.

There's just one problem left. Remember that interrupt that gave a synchronization point between the CPU and the graphics hardware? Turns out

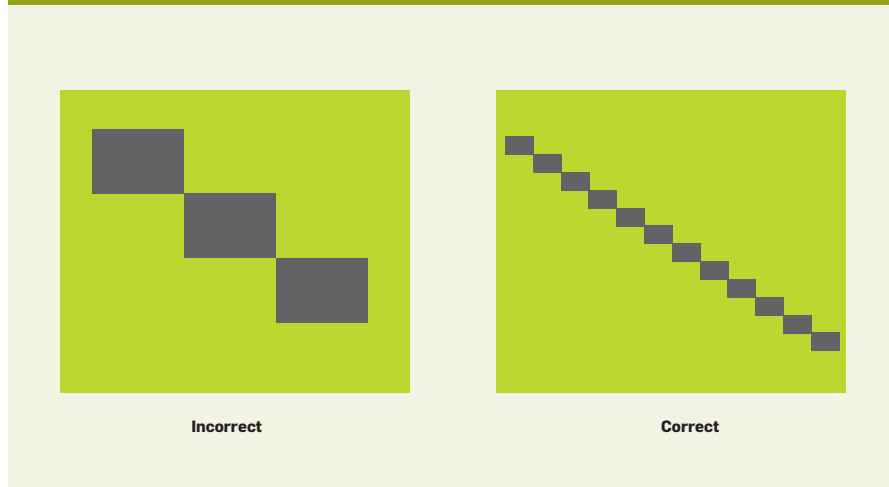
it only happens every *second* frame. The program works but flashes between a perfect diagonal line and a chunky one. There's no hardware facility to help out here, but there is an obvious, if distasteful, software solution.

Once the diagonal line has been drawn, you know exactly when it must be drawn again: 33,792 cycles from when you started drawing it the first time. If it takes  $T$  cycles to draw the line, then you just write a cycle-wasting loop that runs for  $33,792 - T$  cycles and jump back to the line-drawing routine. Since that jump takes 10 cycles, however, you better make that  $33,792 - T - 10$  cycles. This seems like a fine nit to pick, but even being a single cycle off in the count will lose synchronization. In two seconds the sync is off by almost an entire line. Losing sync

has an effect similar to the vertical roll that afflicted old televisions.

An ad hoc solution will work just fine. The proof-of-concept demonstration program will be complete. The possibilities for even more impressive graphics are clear. Hand-timing everything, however, is tedious, slow, and error prone. You're stuck with writing in assembly, but the timing effort takes you back to the days when code was hand-assembled. Much of the burden can be lifted by taking the instruction timing table from the emulator and putting it into the assembler. Assemblers have always been able to measure the size of their output, generally to fill in buffer sizes and the like. Here's that facility in use to define *length* as the number of bytes in a message, which will vary if the message is changed:

Figure 3. A diagonal line in a simple emulator and a more accurate emulator.

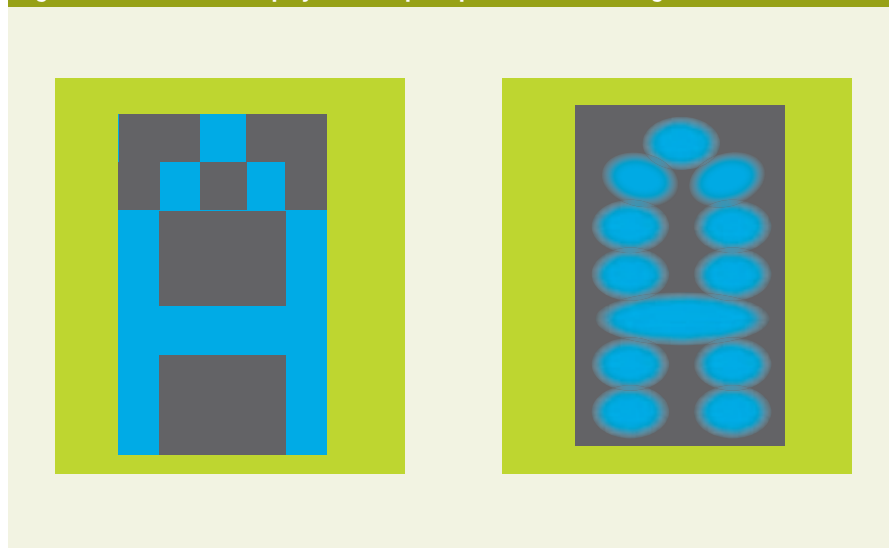


```
message: ascii      'Hello,
world.'
length byte *-message
```

This works because the special "\*" variable keeps track of the memory location into which data and code are assembled. To automate timing simply add a `time()` function that says how many cycles are used by the program up to that point. It can't account for loops and branches but will give accurate results for straight-line code. At a high level the diagonal slash demo will be:

```
start: ...some code to draw
        ...the diagonal line
waste equ      33792      -
(time(*) - time(start)) - 10
        ...code to use up
        ..."waste" cycles
goto start
```

Figure 4. The letter "A" displayed with square pixels and on the original hardware.



Straightforward, but what about the code to waste the cycles? The assembler could be extended to supply that code automatically. Instead, keeping with the principle of minimal design, the task can be left to an ordinary subroutine. Writing a subroutine that runs for a given number of cycles is a different requirement from what you are accustomed to, but it is possible. (See the accompanying sidebar for one such cycle-wasting subroutine.)

As programmers we can see the potential of the diagonal-line demonstration program. Although it has only one pixel per line, there is a clear path to more complex and compelling images, to say nothing of animations and other effects. One final bump in the road awaits. Every time the CPU accesses screen memory, it denies access to the graphics hardware. This results in a blank line that is two- or three-characters wide. The more pixels you change on a per-line basis, the more blanked-out portions there will be. Once again you will find that although the graphics may look fine on the emulator, they will be riddled with “holes” on the real machine because of the blanking side effect.

Moreover, as you try to do more work per line, the exact positions of the blank spots will matter a great deal. Their exact positions will be a measure of emulator accuracy and can be used to maximize the graphics displayed per line. Several discoveries await and will be the result of a feedback loop of emulator refinement, test program development, measurement of the original system leading to further emulator refinement, and so on. Along the way you will discover the following:

- ▶ The visible portion of a line takes 102.4 cycles; the rest of the time (25.6 cycles) is used for setting up drawing the next line.
- ▶ Blank spots do not cover the entire time an instruction takes but only the sub-portion of the instruction that accesses video memory.
- ▶ The emulator must be extended to report exactly when memory is accessed on a sub-instructional basis.
- ▶ Our method of synchronization is crude and can be depended upon to be accurate only to within a few characters.
- ▶ Finer synchronization can be accomplished, but the emulator must be upgraded so programs using the technique can still be tested.
- ▶ Video blanking can be put to good use sculpting graphics that cannot be constructed in other ways.

In other words, we’re a long way from where we started. Instead of drawing an entire screen at once or even a line at a time, the emulator is down to drawing 1/12<sup>th</sup> of a character

at a time and interweaving the CPU and the graphics hardware at the level of CPU cycles. The graphics emulation has become extremely accurate. Not only will side effects such as a tear be seen, but they will be exactly the same as they manifest on the original hardware. The results are not purely academic, either. Test programs demonstrate the fidelity of the emulator while still achieving the same output on the original hardware. The result is not tiny differences only of interest to experts but extremely visible differences in program behavior between precise and sloppy emulators.

Can there be anything else?

Having tripped over so many emulator shortcomings, can the answer be anything but yes? In fact, there is a double-wide mode where the characters are doubled in size for a 32-by-16 display. Based on what we’ve seen up to this point, it’s not surprising to learn that it brings in many more complications than might be expected. Even leaving that morass aside, there’s one more obvious limitation of the emulator. The original display was a CRT. Each pixel on it looks entirely different from what is seen on a modern LCD flat panel. The pixels there are unrelentingly square, whereas the CRT produced soft-edged ovals of phosphorescence. Figure 4 compares two close-ups of the letter A.

Hard-edged pixels result in an image that is functionally identical to the original but has a completely different feel. The difference between the two is unmistakable. Observe also that the real pixels are neither distinct nor independent. Pixels in adjacent rows overlap. Pixels in adjacent columns not only overlap but also display differently if there is a single one versus several in a row. The first pixel in a row of lit pixels is larger. All these subtle differences combine to create a substantially different picture.

The problem itself is much simpler than the functional issues because there is no feedback to the rest of the implementation. There is no need to change the CPU timing or how the CPU interacts with the graphics system. It is merely a matter of drawing each dot as an alpha-blended patch rather than a hard-edged off/on setting of one or two pixels. What is

troublesome is the increased effort required by the host CPU to pull this off. The work involved is many times greater than before. Only through the aid of a graphics coprocessor or moderately optimized rendering code can the screen be drawn in this fashion in real time. It is difficult to believe that drawing a 30-year-old computer’s display takes up so much of a modern system. This is one reason why accurate emulation takes so long to perfect. We can decide to make a better display, but today’s platforms may not have the horsepower to accomplish it.

That realistic fuzzy pixels can overlap does lead to noticeable visual artifacts. Two pixels alternating between on and off sitting side by side will appear to be three pixels: two flashing pixels on each end and a single always-on pixel in the middle where the two overlap. I’ll leave it to your imagination what useful effect this artifact may have.

## Conclusion

A system’s complexity is easy to underestimate. Even the simple video system of the TRS-80 has greater depth than anticipated. What lurks beneath the surface is far greater than the high-level description. Take it as a sideways reinforcement of the KISS principle. Yet do not despair. You must also consider the power of tools. Each emulator improvement has led to discoveries that could be exploited for good use once the necessary support tools were built. Above all, however, beware of perfection. No system is perfect, and the cost of pursuing perfection can be much greater than mere time and money invested. ■

## Related articles on [queue.acm.org](http://queue.acm.org)

### No Source Code? No Problem!

Peter Phillips and George Phillips  
<http://queue.acm.org/detail.cfm?id=945155>

### Enhanced Debugging with Traces

Peter Phillips  
<http://queue.acm.org/detail.cfm?id=1753170>

George Phillips (gp2000@shaw.ca) works on video games both emulated and otherwise. In a simpler time he worked on the crawling side of an early Web search engine that was chastised for selling advertisements.

© 2010 ACM 0001-0782/10/0600 \$10.00

---

**A survey of powerful visualization techniques,  
from the obvious to the obscure.**

---

**BY JEFFREY HEER, MICHAEL BOSTOCK, AND VADIM OGIEVETSKY**

---

# A Tour Through the Visualization Zoo

THANKS TO ADVANCES in sensing, networking, and data management, our society is producing digital information at an astonishing rate. According to one estimate, in 2010 alone we will generate 1,200 exabytes—60 million times the content of the Library of Congress. Within this deluge of data lies a wealth

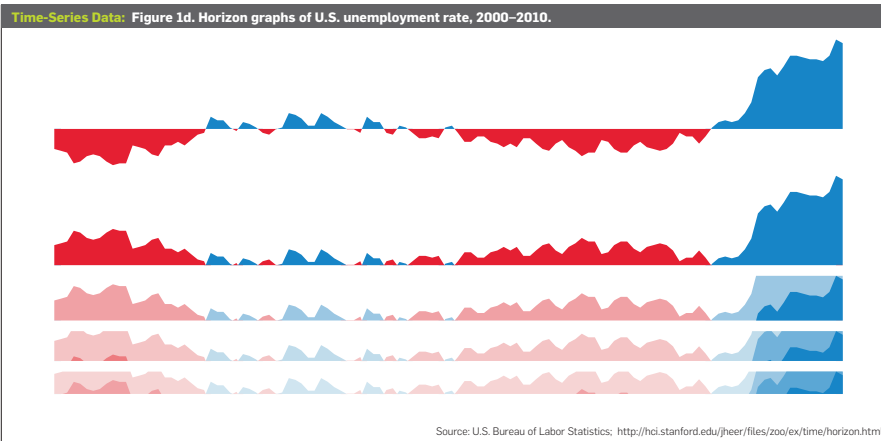
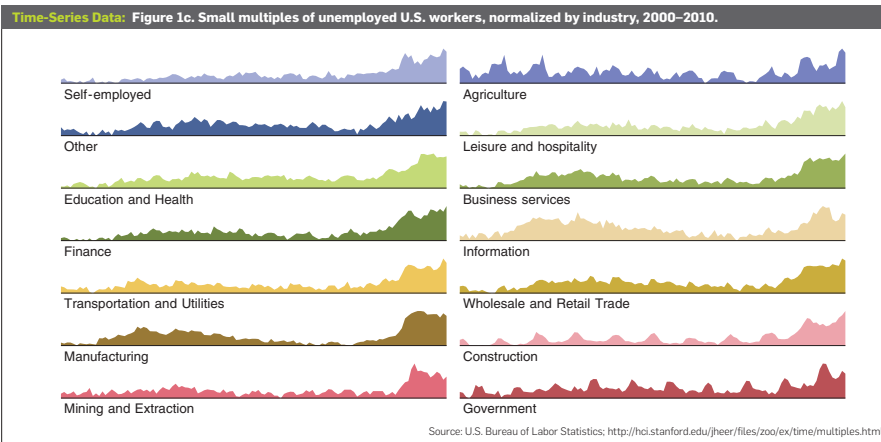
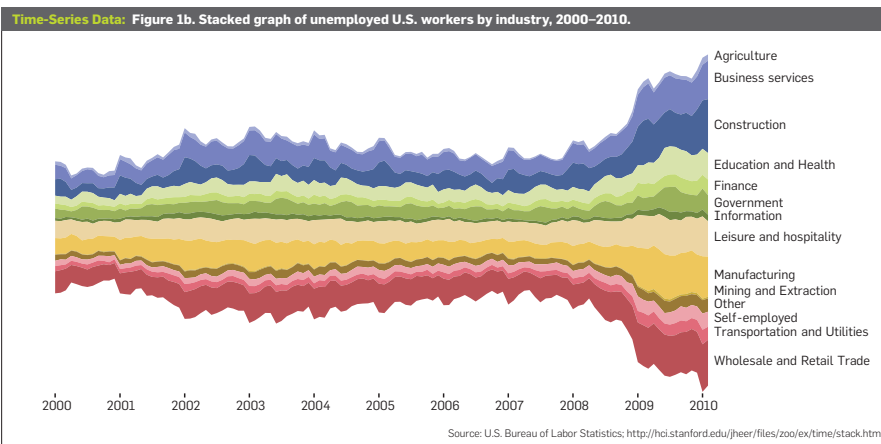
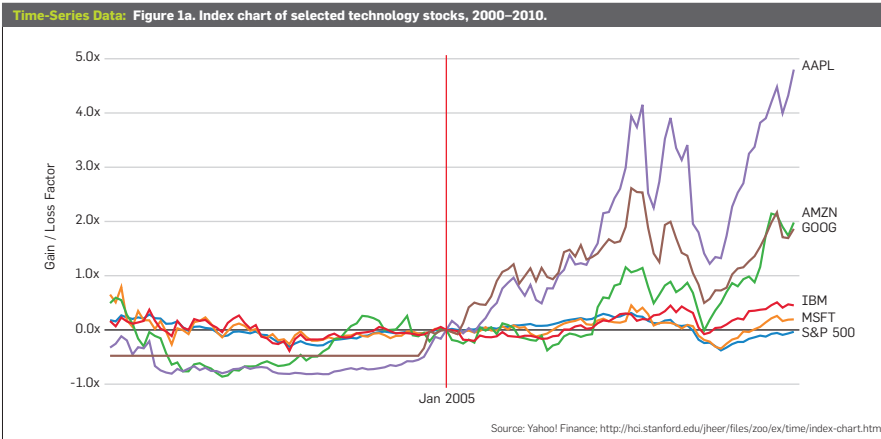
of valuable information on how we conduct our businesses, governments, and personal lives. To put the information to good use, we must find ways to explore, relate, and communicate the data meaningfully.

The goal of visualization is to aid our understanding of data by leveraging the human visual system's highly tuned ability to see patterns, spot trends, and identify outliers. Well-designed visual representations can replace cognitive calculations with simple perceptual inferences and improve comprehension, memory, and decision making. By making data more accessible and appealing, visual representations may also

help engage more diverse audiences in exploration and analysis. The challenge is to create effective and engaging visualizations that are appropriate to the data.

Creating a visualization requires a number of nuanced judgments. One must determine which questions to ask, identify the appropriate data, and select effective *visual encodings* to map data values to graphical features such as position, size, shape, and color. The challenge is that for any given data set the number of visual encodings—and thus the space of possible visualization designs—is extremely large. To guide this process, computer scientists, psy-





chologists, and statisticians have studied how well different encodings facilitate the comprehension of data types such as numbers, categories, and networks. For example, *graphical perception* experiments find that spatial position (as in a scatter plot or bar chart) leads to the most accurate decoding of numerical data and is generally preferable to visual variables such as angle, one-dimensional length, two-dimensional area, three-dimensional volume, and color saturation. Thus, it should be no surprise that the most common data graphics, including bar charts, line charts, and scatter plots, use position encodings. Our understanding of graphical perception remains incomplete, however, and must appropriately be balanced with interaction design and aesthetics.

This article provides a brief tour through the “visualization zoo,” showcasing techniques for visualizing and interacting with diverse data sets. In many situations, simple data graphics will not only suffice, they may also be preferable. Here we focus on a few of the more sophisticated and unusual techniques that deal with complex data sets. After all, you don’t go to the zoo to see chihuahuas and raccoons; you go to admire the majestic polar bear, the graceful zebra, and the terrifying Sumatran tiger. Analogously, we cover some of the more exotic (but practically useful) forms of visual data representation, starting with one of the most common, time-series data; continuing on to statistical data and maps; and then completing the tour with hierarchies and networks. Along the way, bear in mind that all visualizations share a common “DNA”—a set of mappings between data properties and visual attributes such as position, size, shape, and color—and that customized species of visualization might always be constructed by varying these encodings.

Each visualization shown here is accompanied by an online interactive example that can be viewed at the URL displayed beneath it. The live examples were created using *Protovis*, an open source language for Web-based data visualization. To learn more about how a visualization was made (or to copy and paste it for your own use), see the online version of this article available on the *ACM Queue* site at <http://queue>.

acm.org/detail.cfm?id=1780401/. All example source code is released into the public domain and has no restrictions on reuse or modification. Note, however, that these examples will work only on a modern, standards-compliant browser supporting scalable vector graphics (SVG). Supported browsers include recent versions of Firefox, Safari, Chrome, and Opera. Unfortunately, Internet Explorer 8 and earlier versions do not support SVG and so cannot be used to view the interactive examples.

### Time-Series Data

Sets of values changing over time—or, time-series data—is one of the most common forms of recorded data. Time-varying phenomena are central to many domains such as finance (stock prices, exchange rates), science (temperatures, pollution levels, electric potentials), and public policy (crime rates). One often needs to compare a large number of time series simultaneously and can choose from a number of visualizations to do so.

**Index Charts.** With some forms of time-series data, raw values are less important than relative changes. Consider investors who are more interested in a stock's growth rate than its specific price. Multiple stocks may have dramatically different baseline prices but may be meaningfully compared when normalized. An *index chart* is an interactive line chart that shows percentage changes for a collection of time-series data based on a selected index point. For example, the image in Figure 1a shows the percentage change of selected stock prices if purchased in January 2005: one can see the rocky rise enjoyed by those who invested in Amazon, Apple, or Google at that time.

**Stacked Graphs.** Other forms of time-series data may be better seen in aggregate. By stacking area charts on top of each other, we arrive at a visual summation of time-series values—a *stacked graph*. This type of graph (sometimes called a *stream graph*) depicts aggregate patterns and often supports drill-down into a subset of individual series. The chart in Figure 1b shows the number of unemployed workers in the U.S. over the past decade, subdivided by industry. While such charts have proven popular in recent years, they do have some notable limitations. A stacked

graph does not support negative numbers and is meaningless for data that should not be summed (temperatures, for example). Moreover, stacking may make it difficult to accurately interpret trends that lie atop other curves. Interactive search and filtering is often used to compensate for this problem.

**Small Multiples.** In lieu of stacking, multiple time series can be plotted within the same axes, as in the index chart. Placing multiple series in the same space may produce overlapping curves that reduce legibility, however. An alternative approach is to use *small multiples*: showing each series in its own chart. In Figure 1c we again see the number of unemployed workers, but normalized within each industry category. We can now more accurately see both overall trends and seasonal patterns in each sector. While we are considering time-series data, note that small multiples can be constructed for just about any type of visualization: bar charts, pie charts, maps, among others. This often produces a more effective visualization than trying to coerce all the data into a single plot.

**Horizon Graphs.** What happens when you want to compare even more time series at once? The *horizon graph* is a technique for increasing the data density of a time-series view while preserving resolution. Consider the five graphs shown in Figure 1d. The first one is a standard area chart, with positive values colored blue and negative values colored red. The second graph “mirrors” negative values into the same region as positive values, doubling the data density of the area chart. The third chart—a horizon graph—doubles the data density yet again by dividing the graph into bands and layering them to create a nested form. The result is a chart that preserves data resolution but uses only a quarter of the space. Although the horizon graph takes some time to learn, it has been found to be more effective than the standard plot when the chart sizes get quite small.

### Statistical Distributions

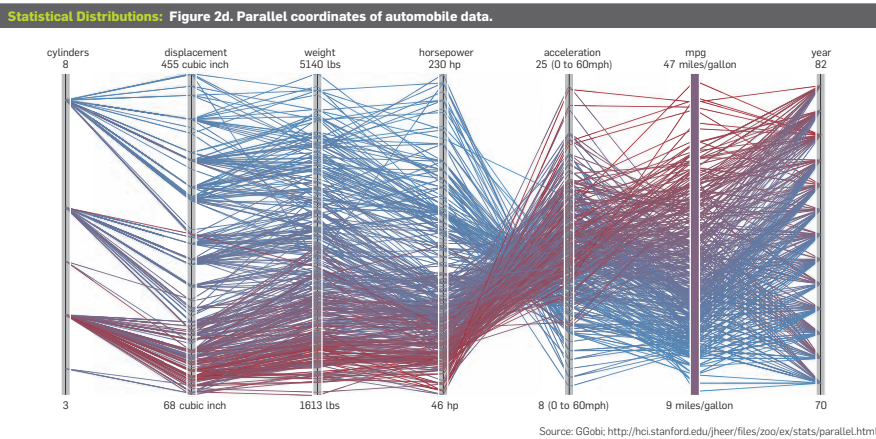
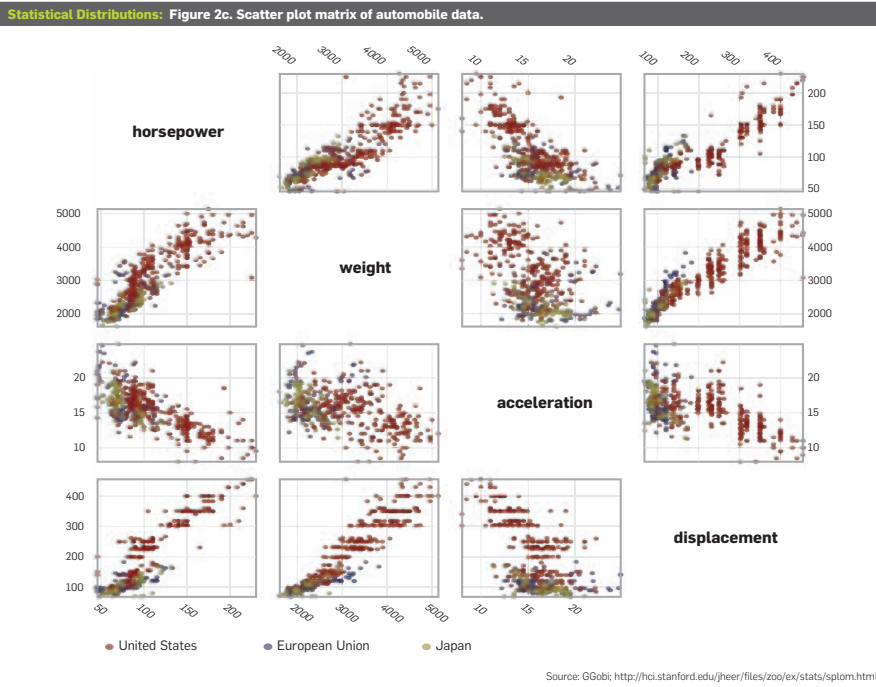
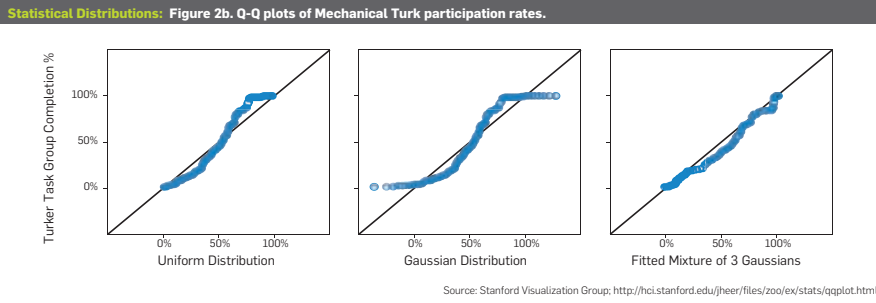
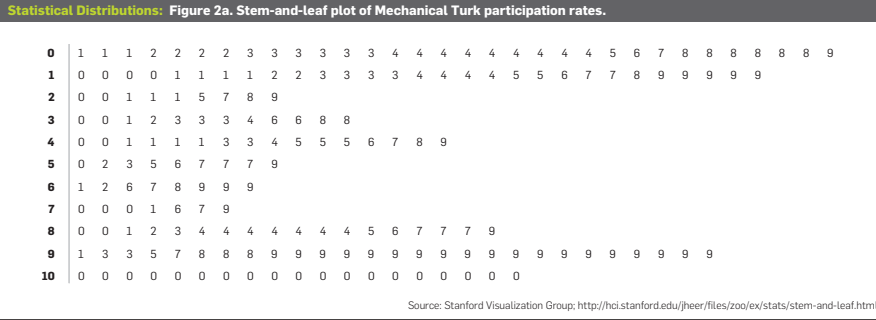
Other visualizations have been designed to reveal how a set of numbers is distributed and thus help an analyst better understand the statistical properties of the data. Analysts often want to fit their data to statistical models, ei-

ther to test hypotheses or predict future values, but an improper choice of model can lead to faulty predictions. Thus, one important use of visualizations is *exploratory data analysis*: gaining insight into how data is distributed to inform data transformation and modeling decisions. Common techniques include the *histogram*, which shows the prevalence of values grouped into bins, and the *box-and-whisker plot*, which can convey statistical features such as the mean, median, quartile boundaries, or extreme outliers. In addition, a number of other techniques exist for assessing a distribution and examining interactions between multiple dimensions.

**Stem-and-Leaf Plots.** For assessing a collection of numbers, one alternative to the histogram is the *stem-and-leaf plot*. It typically bins numbers according to the first significant digit, and then stacks the values within each bin by the second significant digit. This minimalistic representation uses the data itself to paint a frequency distribution, replacing the “information-empty” bars of a traditional histogram bar chart and allowing one to assess both the overall distribution and the contents of each bin. In Figure 2a, the stem-and-leaf plot shows the distribution of completion rates of workers completing crowd-sourced tasks on Amazon's Mechanical Turk. Note the multiple clusters: one group clusters around high levels of completion (99%–100%); at the other extreme is a cluster of Turkers who complete only a few tasks (~10%) in a group.

**Q-Q Plots.** Though the histogram and the stem-and-leaf plot are common tools for assessing a frequency distribution, the *Q-Q (quantile-quantile)* plot is a more powerful tool. The Q-Q plot compares two probability distributions by graphing their quantiles against each other. If the two are similar, the plotted values will lie roughly along the central diagonal. If the two are linearly related, values will again lie along a line, though with varying slope and intercept.

Figure 2b shows the same Mechanical Turk participation data compared with three statistical distributions. Note how the data forms three distinct components when compared with uniform and normal (Gaussian) distributions: this suggests that a statistical model with three components might



be more appropriate, and indeed we see in the final plot that a fitted mixture of three normal distributions provides a better fit. Though powerful, the Q-Q plot has one obvious limitation in that its effective use requires that viewers possess some statistical knowledge.

**SPLOM (Scatter Plot Matrix).** Other visualization techniques attempt to represent the relationships among multiple variables. Multivariate data occurs frequently and is notoriously hard to represent, in part because of the difficulty of mentally picturing data in more than three dimensions. One technique to overcome this problem is to use small multiples of scatter plots showing a set of pairwise relations among variables, thus creating the *SPLOM (scatter plot matrix)*. A SPLOM enables visual inspection of correlations between any pair of variables.

In Figure 2c a scatter plot matrix is used to visualize the attributes of a database of automobiles, showing the relationships among horsepower, weight, acceleration, and displacement. Additionally, interaction techniques such as *brushing-and-linking*—in which a selection of points on one graph highlights the same points on all the other graphs—can be used to explore patterns within the data.

**Parallel Coordinates.** As shown in Figure 2d, parallel coordinates (*||-cord*) take a different approach to visualizing multivariate data. Instead of graphing every pair of variables in two dimensions, we repeatedly plot the data on parallel axes and then connect the corresponding points with lines. Each poly-line represents a single row in the database, and line crossings between dimensions often indicate inverse correlation. Reordering dimensions can aid pattern-finding, as can interactive querying to filter along one or more dimensions. Another advantage of parallel coordinates is that they are relatively compact, so many variables can be shown simultaneously.

**Maps**

Although a map may seem a natural way to visualize geographical data, it has a long and rich history of design. Many maps are based upon a *cartographic projection*: a mathematical function that maps the 3D geometry of the Earth to a 2D image. Other maps



knowingly distort or abstract geographic features to tell a richer story or highlight specific data.

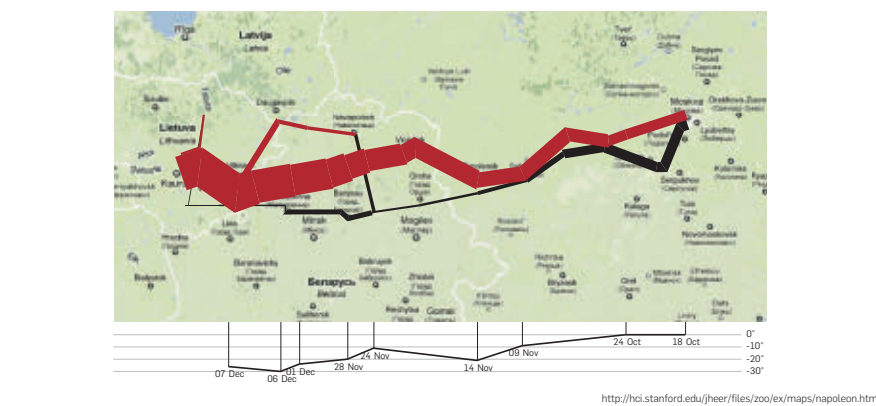
**Flow Maps.** By placing stroked lines on top of a geographic map, a *flow map* can depict the movement of a quantity in space and (implicitly) in time. Flow lines typically encode a large amount of multivariate information: path points, direction, line thickness, and color can all be used to present dimensions of information to the viewer. Figure 3a is a modern interpretation of Charles Minard's depiction of Napoleon's ill-fated march on Moscow. Many of the greatest flow maps also involve subtle uses of distortion, as geography is bended to accommodate or highlight flows.

**Choropleth Maps.** Data is often collected and aggregated by geographical areas such as states. A standard approach to communicating this data is to use a color encoding of the geographic area, resulting in a *choropleth map*. Figure 3b uses a color encoding to communicate the prevalence of obesity in each state in the U.S. Though this is a widely used visualization technique, it requires some care. One common error is to encode raw data values (such as population) rather than using normalized values to produce a density map. Another issue is that one's perception of the shaded value can also be affected by the underlying area of the geographic region.

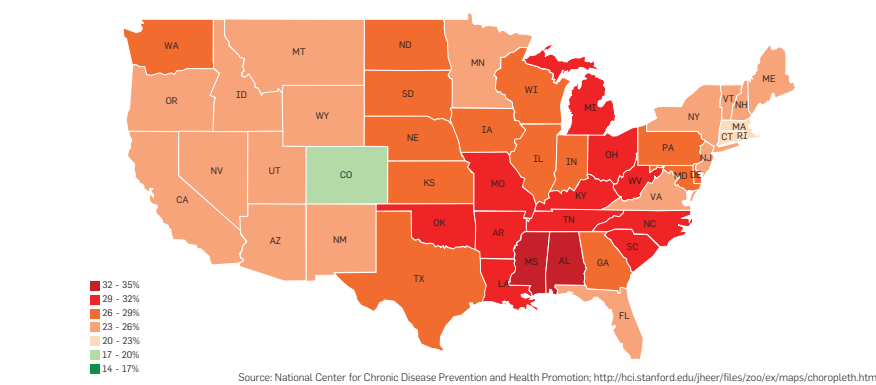
**Graduated Symbol Maps.** An alternative to the choropleth map, the *graduated symbol map* places symbols over an underlying map. This approach avoids confounding geographic area with data values and allows for more dimensions to be visualized (for example, symbol size, shape, and color). In addition to simple shapes such as circles, graduated symbol maps may use more complicated glyphs such as pie charts. In Figure 3c, total circle size represents a state's population, and each slice indicates the proportion of people with a specific BMI rating.

**Cartograms.** A *cartogram* distorts the shape of geographic regions so that the area directly encodes a data variable. A common example is to redraw every country in the world sizing it proportionally to population or gross domestic product. Many types of cartograms have been created; in Figure 3d we use the *Dorling cartogram*, which represents

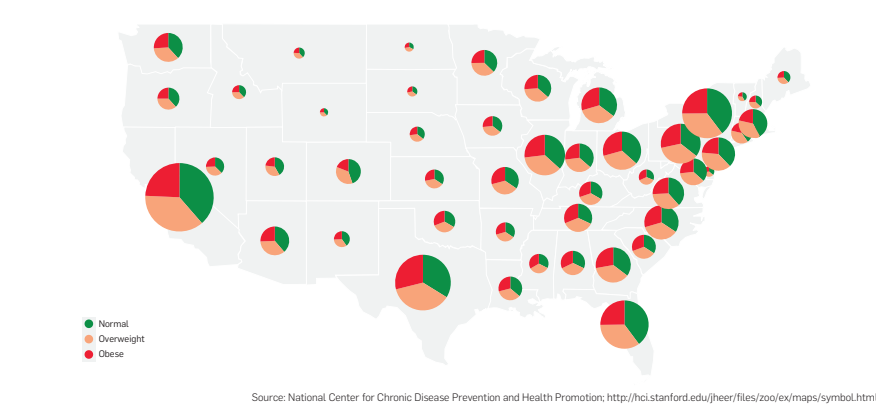
Maps: Figure 3a. Flow map of Napoleon's March on Moscow, based on the work of Charles Minard.



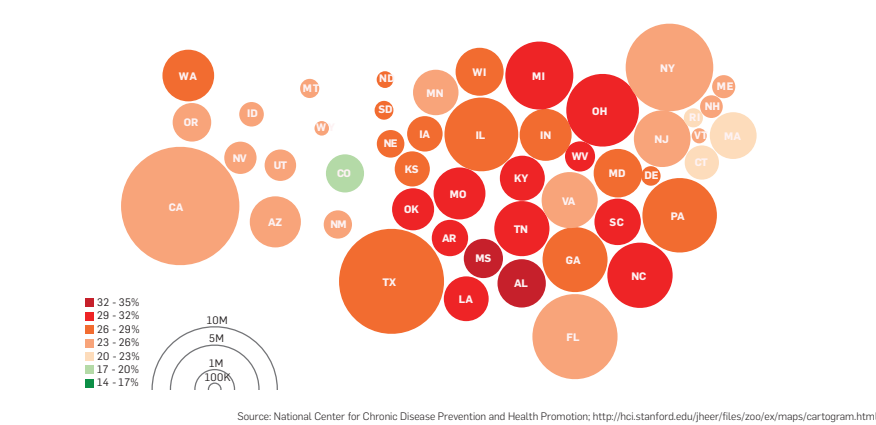
Maps: Figure 3b. Choropleth map of obesity in the U.S., 2008.



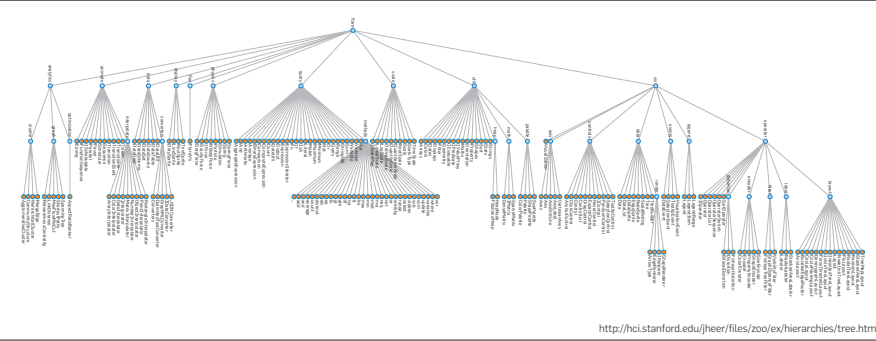
Maps: Figure 3c. Graduated symbol map of obesity in the U.S., 2008.



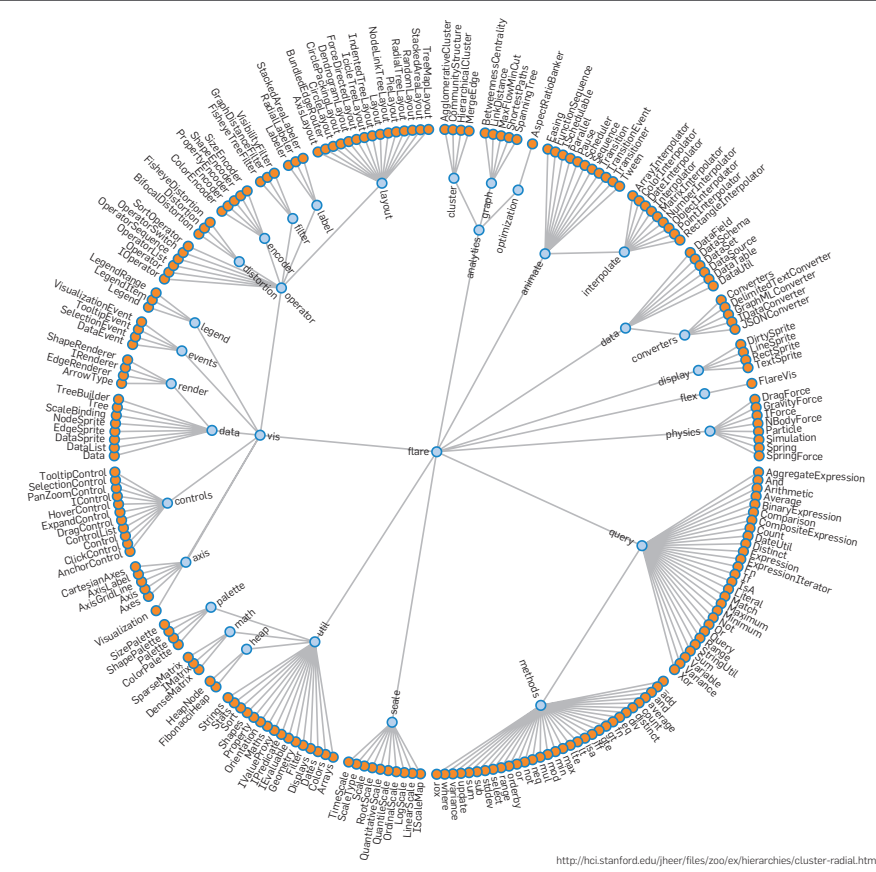
Maps: Figure 3d. Dorling cartogram of obesity in the U.S., 2008.



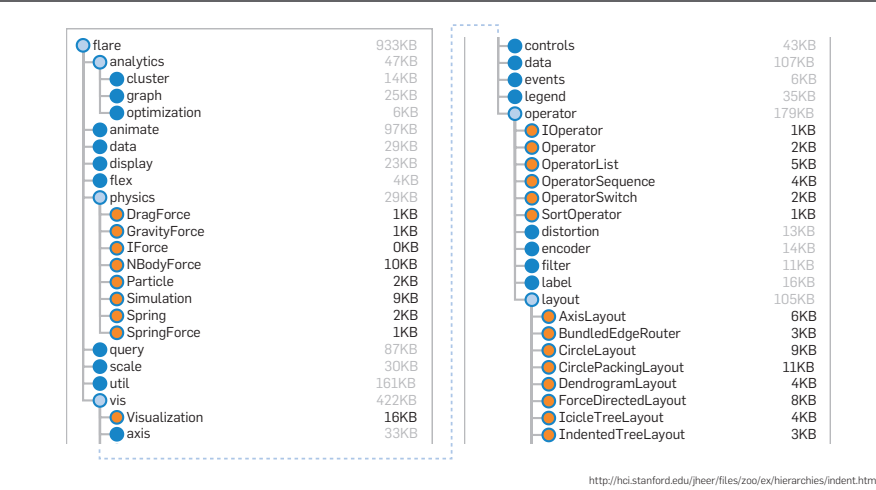
Hierarchies: Figure 4a. Radial node-link diagram of the Flare package hierarchy.



Hierarchies: Figure 4b. Cartesian node-link diagram of the Flare package hierarchy.



Hierarchies: Figure 4c. Indented tree layout of the Flare package hierarchy.



each geographic region with a sized circle, placed so as to resemble the true geographic configuration. In this example, circular area encodes the total number of obese people per state, and color encodes the percentage of the total population that is obese.

### Hierarchies

While some data is simply a flat collection of numbers, most can be organized into natural hierarchies. Consider: spatial entities, such as counties, states, and countries; command structures for businesses and governments; software packages and phylogenetic trees. Even for data with no apparent hierarchy, statistical methods (for example, *k-means clustering*) may be applied to organize data empirically. Special visualization techniques exist to leverage hierarchical structure, allowing rapid multiscale inferences: micro-observations of individual elements and macro-observations of large groups.

**Node-link diagrams.** The word *tree* is used interchangeably with *hierarchy*, as the fractal branches of an oak might mirror the nesting of data. If we take a two-dimensional blueprint of a tree, we have a popular choice for visualizing hierarchies: a *node-link diagram*. Many different tree-layout algorithms have been designed; the Reingold-Tilford algorithm, used in Figure 4a on a package hierarchy of software classes, produces a tidy result with minimal wasted space.

An alternative visualization scheme is the *dendrogram* (or cluster) algorithm, which places leaf nodes of the tree at the same level. Thus, in the diagram in Figure 4b, the classes (orange leaf nodes) are on the diameter of the circle, with the packages (blue internal nodes) inside. Using polar rather than Cartesian coordinates has a pleasing aesthetic, while using space more efficiently.

We would be remiss to overlook the indented tree, used ubiquitously by operating systems to represent file directories, among other applications (see Figure 4c). Although the indented tree requires excessive vertical space and does not facilitate multiscale inferences, it does allow efficient *interactive* exploration of the tree to find a specific node. In addition, it allows rapid scanning of node labels, and multivariate data such as file size can be displayed adjacent to the hierarchy.

**Adjacency Diagrams.** The *adjacency diagram* is a space-filling variant of the node-link diagram; rather than drawing a link between parent and child in the hierarchy, nodes are drawn as solid areas (either arcs or bars), and their placement relative to adjacent nodes reveals their position in the hierarchy. The icicle layout in Figure 4d is similar to the first node-link diagram in that the root node appears at the top, with child nodes underneath. Because the nodes are now space-filling, however, we can use a length encoding for the size of software classes and packages. This reveals an additional dimension that would be difficult to show in a node-link diagram.

The sunburst layout, shown in Figure 4e, is equivalent to the icicle layout, but in polar coordinates. Both are implemented using a partition layout, which can also generate a node-link diagram. Similarly, the previous cluster layout can be used to generate a space-filling adjacency diagram in either Cartesian or polar coordinates.

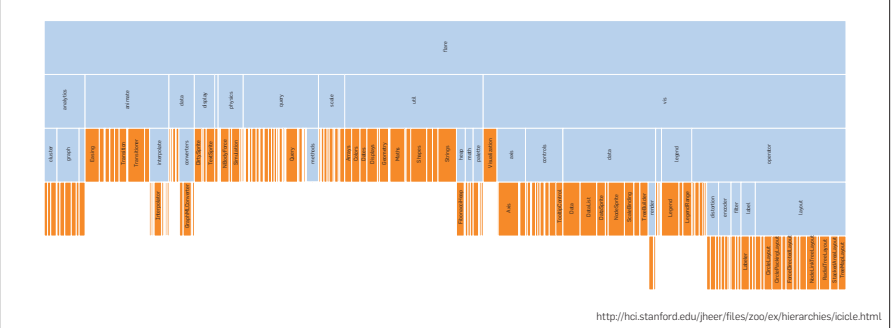
**Enclosure Diagrams.** The *enclosure diagram* is also space filling, using containment rather than adjacency to represent the hierarchy. Introduced by Ben Shneiderman in 1991, a *treemap* recursively subdivides area into rectangles. As with adjacency diagrams, the size of any node in the tree is quickly revealed. The example shown in Figure 4f uses padding (in blue) to emphasize enclosure; an alternative saturation encoding is sometimes used. *Squarified* treemaps use approximately square rectangles, which offer better readability and size estimation than a naive “slice-and-dice” subdivision. Fancier algorithms such as Voronoi and jigsaw treemaps also exist but are less common.

By packing circles instead of subdividing rectangles, we can produce a different sort of enclosure diagram that has an almost organic appearance. Although it does not use space as efficiently as a treemap, the “wasted space” of the *circle-packing layout*, shown in Figure 4g, effectively reveals the hierarchy. At the same time, node sizes can be rapidly compared using area judgments.

**Networks**

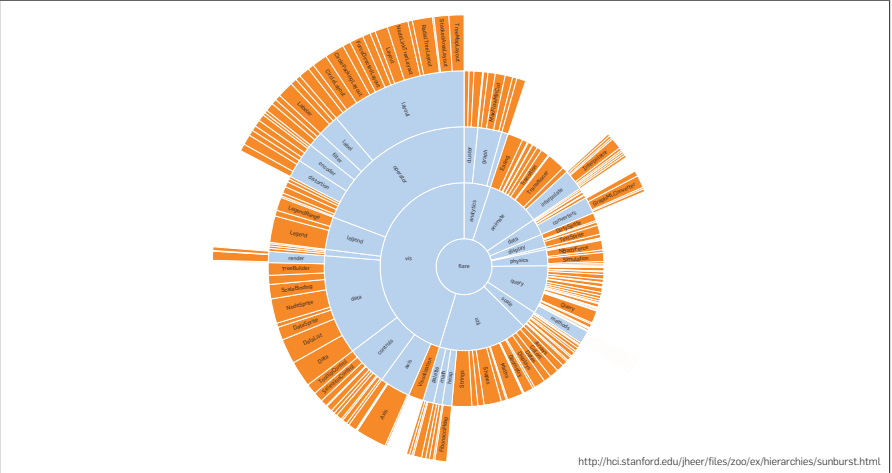
In addition to organization, one aspect

Hierarchies: Figure 4d. Icicle tree layout of the Flare package hierarchy.



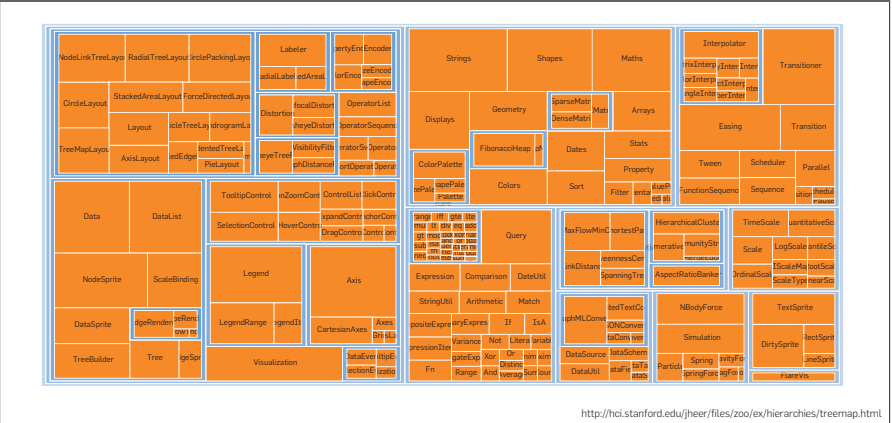
<http://hci.stanford.edu/jheer/files/zoo/ex/hierarchies/icicle.html>

Hierarchies: Figure 4e. Sunburst (radial space-filling) layout of the Flare package hierarchy.



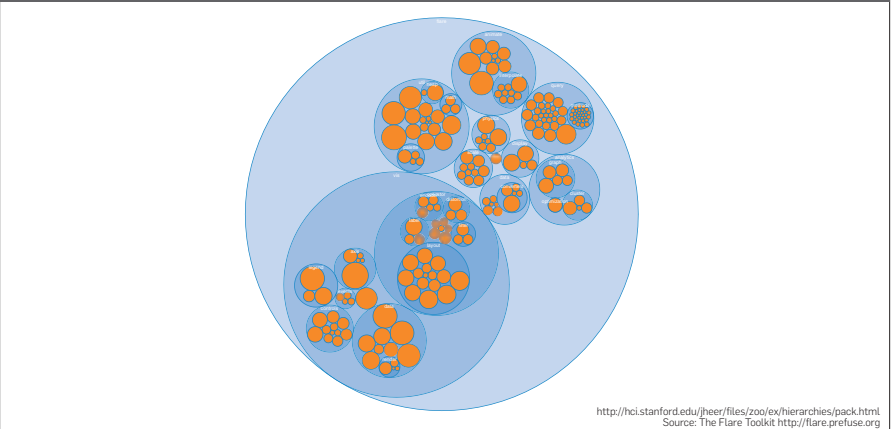
<http://hci.stanford.edu/jheer/files/zoo/ex/hierarchies/sunburst.html>

Hierarchies: Figure 4f. Treemap layout of the Flare package hierarchy.



<http://hci.stanford.edu/jheer/files/zoo/ex/hierarchies/treemap.html>

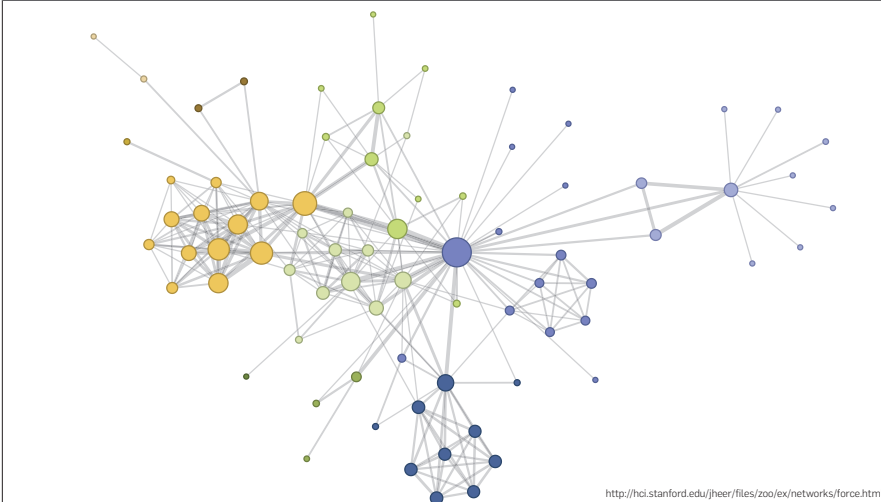
Hierarchies: Figure 4g. Nested circles layout of the Flare package hierarchy.



<http://hci.stanford.edu/jheer/files/zoo/ex/hierarchies/pack.html>  
Source: The Flare Toolkit <http://flare.prefuse.org>

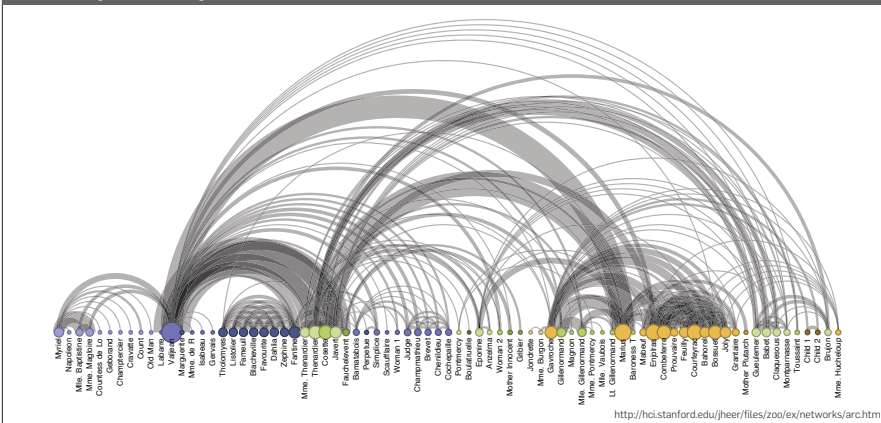


Networks: Figure 5a. Force-directed layout of *Les Misérables* character co-occurrences.



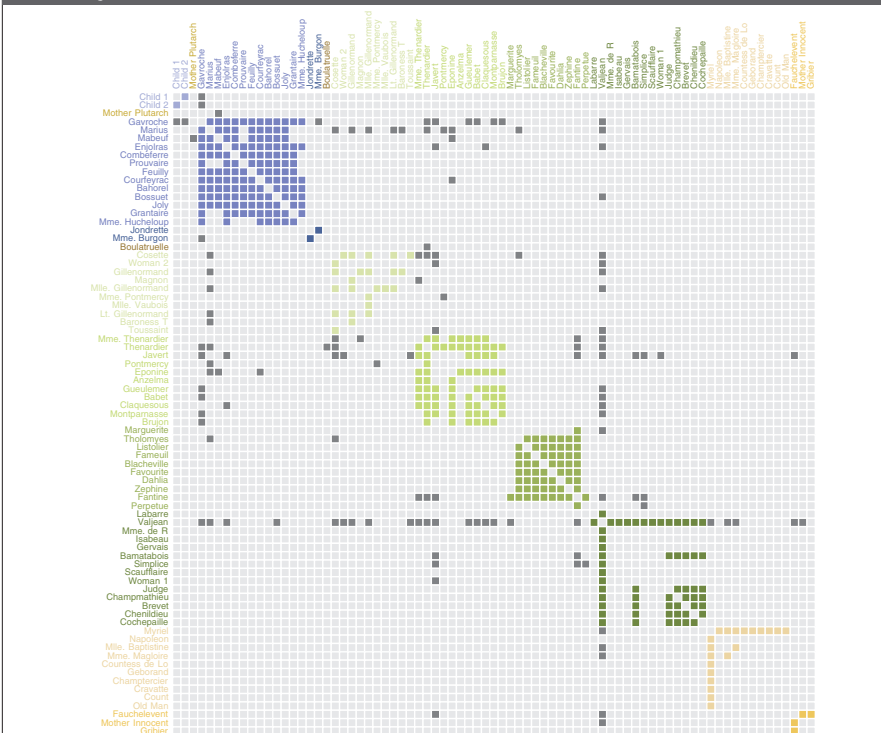
<http://hci.stanford.edu/jheer/files/zoo/ex/networks/force.html>

Networks: Figure 5b. Arc diagram of *Les Misérables* character co-occurrences.



<http://hci.stanford.edu/jheer/files/zoo/ex/networks/arc.html>

Networks: Figure 5c. Matrix view of *Les Misérables* character co-occurrences.



<http://hci.stanford.edu/jheer/files/zoo/ex/networks/matrix.html>  
Source: <http://www-personal.umich.edu/~mejn/netdata>

of data that we may wish to explore through visualization is relationship. For example, given a social network, who is friends with whom? Who are the central players? What cliques exist? Who, if anyone, serves as a bridge between disparate groups? Abstractly, a hierarchy is a specialized form of network: each node has exactly one link to its parent, while the root node has no links. Thus node-link diagrams are also used to visualize networks, but the loss of hierarchy means a different algorithm is required to position nodes.

Mathematicians use the formal term *graph* to describe a network. A central challenge in graph visualization is computing an effective layout. Layout techniques typically seek to position closely related nodes (in terms of *graph distance*, such as the number of links between nodes, or other metrics) close in the drawing; critically, *unrelated* nodes must also be placed far enough apart to differentiate relationships. Some techniques may seek to optimize other visual features—for example, by minimizing the number of edge crossings.

**Force-directed Layouts.** A common and intuitive approach to network layout is to model the graph as a physical system: nodes are charged particles that repel each other, and links are dampened springs that pull related nodes together. A physical simulation of these forces then determines the node positions; approximation techniques that avoid computing all pairwise forces enable the layout of large numbers of nodes. In addition, interactivity allows the user to direct the layout and jiggle nodes to disambiguate links. Such a *force-directed layout* is a good starting point for understanding the structure of a general undirected graph. In Figure 5a we use a force-directed layout to view the network of character co-occurrence in the chapters of Victor Hugo’s classic novel, *Les Misérables*. Node colors depict cluster memberships computed by a community-detection algorithm.

**Arc Diagrams.** An *arc diagram*, shown in Figure 5b, uses a one-dimensional layout of nodes, with circular arcs to represent links. Though an arc diagram may not convey the overall structure of the graph as effectively as a two-dimensional layout, with a good ordering of nodes it is easy to identify

cliques and bridges. Further, as with the indented-tree layout, multivariate data can easily be displayed alongside nodes. The problem of sorting the nodes in a manner that reveals underlying cluster structure is formally called *seriation* and has diverse applications in visualization, statistics, and even archaeology.


**Matrix Views.** Mathematicians and computer scientists often think of a graph in terms of its *adjacency matrix*: each value in row  $i$  and column  $j$  in the matrix corresponds to the link from node  $i$  to node  $j$ . Given this representation, an obvious visualization then is: just show the matrix! Using color or saturation instead of text allows values associated with the links to be perceived more rapidly.

The seriation problem applies just as much to the *matrix view*, shown in Figure 5c, as to the arc diagram, so the order of rows and columns is important: here we use the groupings generated by a community-detection algorithm to order the display. While path-following is more difficult in a matrix view than in a node-link diagram, matrices have a number of compensating advantages. As networks get large and highly connected, node-link diagrams often devolve into giant hairballs of line crossings. In matrix views, however, line crossings are impossible, and with an effective sorting one quickly can spot clusters and bridges. Allowing interactive grouping and reordering of the matrix facilitates even deeper exploration of network structure.


## Conclusion


We have arrived at the end of our tour and hope the reader has found the examples both intriguing and practical. Though we have visited a number of visual encoding and interaction techniques, many more species of visualization exist in the wild, and others await discovery. Emerging domains such as bioinformatics and text visualization are driving researchers and designers to continually formulate new and creative representations or find more powerful ways to apply the classics. In either case, the DNA underlying all visualizations remains the same: the principled mapping of data variables to visual features such as position, size, shape, and color.

As you leave the zoo and head back



**All visualizations share a common “DNA”—a set of mappings between data properties and visual attributes such as position, size, shape, and color—and customized species of visualization might always be constructed by varying these encodings.**



into the wild, try deconstructing the various visualizations crossing your path. Perhaps you can design a more effective display? 

### Additional Resources

Few, S.

*Now I See It: Simple Visualization Techniques for Quantitative Analysis.* Analytics Press, 2009.

Tufte, E.

*The Visual Display of Quantitative Information.* Graphics Press, 1983.

Tufte, E.

*Envisioning Information.* Graphics Press, 1990.

Ware, C.

*Visual Thinking for Design.* Morgan Kaufmann, 2008.

Wilkinson, L.

*The Grammar of Graphics.* Springer, 1999.

### Visualization Development Tools

*Prefuse:* Java API for information visualization.

*Prefuse Flare:* ActionScript 3 library for data visualization in the Adobe Flash Player.

*Processing:* Popular language and IDE for graphics and interaction.

*Protovis:* JavaScript tool for Web-based visualization.

*The Visualization Toolkit:* Library for 3D and scientific visualization.

### Related articles on [queue.acm.org](http://queue.acm.org)

**A Conversation with Jeff Heer, Martin Wattenberg, and Fernanda Viégas**

<http://queue.acm.org/detail.cfm?id=1744741>

**Unifying Biological Image Formats with HDF5**

Matthew T. Dougherty, Michael J. Folk, Erez Zadok, Herbert J. Bernstein, Frances C. Bernstein, Kevin W. Eliceiri, Werner Benger, Christoph Best

<http://queue.acm.org/detail.cfm?id=1628215>

**Jeffrey Heer** is an assistant professor of computer science at Stanford University, where he works on human-computer interaction, visualization, and social computing. He led the design of the Prefuse, Flare, and Protovis visualization toolkits.

**Michael Bostock** is currently a Ph.D. student in the Department of Computer Science at Stanford University. Before attending Stanford, he was a staff engineer at Google, where he developed search quality evaluation methodologies.

**Vadim Ogievetsky** is a master's student at Stanford University specializing in human-computer interaction. He is a core contributor to Protovis, an open-source Web-based visualization toolkit.

DOI:10.1145/1743546.1743568

**Needed are generic, rather than one-off, DBMS solutions automating storage and analysis of data from scientific collaborations.**

**BY ANASTASIA AILAMAKI, VERENA KANTERE,  
AND DEBABRATA DASH**

## Managing Scientific Data

DATA-ORIENTED SCIENTIFIC PROCESSES depend on fast, accurate analysis of experimental data generated through empirical observation and simulation. However, scientists are increasingly overwhelmed by the volume of data produced by their own experiments. With improving instrument precision and the complexity of the simulated models, data overload promises to only get worse. The inefficiency of existing database management systems (DBMSs) for addressing the requirements of scientists has led to many application-specific systems. Unlike their general-purpose counterparts, these systems require more resources, hindering reuse of knowledge. Still, the data-management community aspires to general-purpose scientific data management. Here, we explore the most important requirements of such systems and the techniques being used to address them.

Observation and simulation of phenomena are keys for proving scientific theories and discovering facts of

nature the human brain could otherwise never imagine. Scientists must be able to manage data derived from observations and simulations. Constant improvement of observational instruments and simulation tools give modern science effective options for abundant information capture, reflecting the rich diversity of complex life forms and cosmic phenomena. Moreover, the need for in-depth analysis of huge amounts of data relentlessly drives demand for additional computational support.

Microsoft researcher and ACM Turing Award laureate Jim Gray once said, “A fourth data-intensive science is emerging. The goal is to have a world in which all of the science literature is online, all the science data is online, and they interoperate with each other.”<sup>9</sup> Unfortunately, today’s commercial data-management tools are incapable of supporting the unprecedented scale, rate, and complexity of scientific data collection and processing.

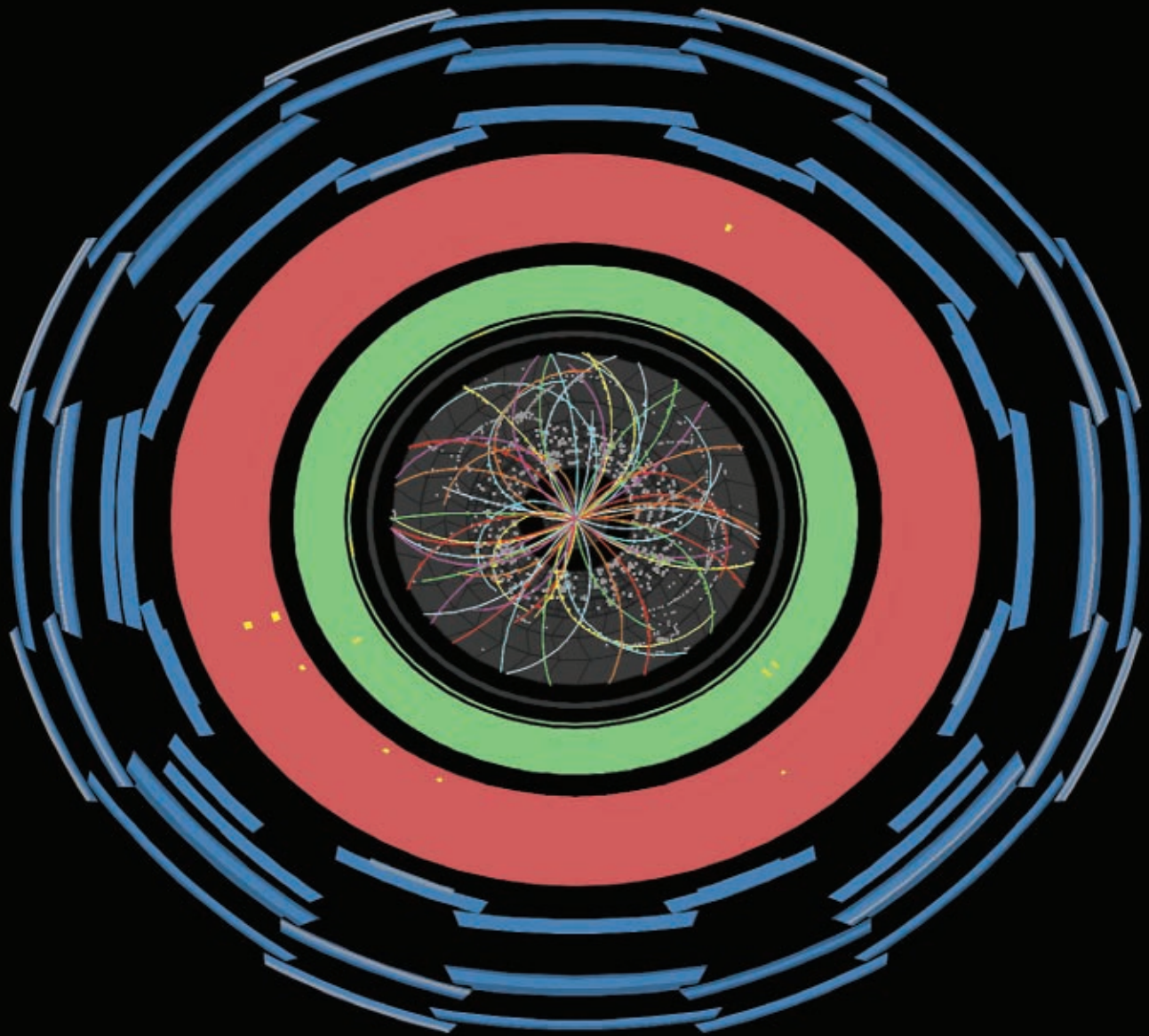
Despite its variety, scientific data does share some common features:

- ▶ Scale usually dwarfing the scale of transactional data sets;
- ▶ Generated through complex and interdependent workflows;
- ▶ Typically multidimensional;
- ▶ Embedded physical models;
- ▶ Important metadata about experiments and their provenance;
- ▶ Floating-point heavy; and
- ▶ Low update rates, with most updates append-only.

### » key insights

- **Managing the enormous amount of scientific data being collected is the key to scientific progress.**
- **Though technology allows for the extreme collection rates of scientific data, processing is still performed with stale techniques developed for small data sets; efficient processing is necessary to be able to exploit the value of huge scientific data collections.**
- **Proposed solutions also promise to achieve efficient management for almost any other kind of data.**





**Result of seven-trillion-electronvolt collisions (March 30, 2010) in the ATLAS particle detector on the Large Hadron Collider at CERN, hunting for dark matter, new forces, new dimensions, the Higgs boson, and ultimately a grand theory to explain all physical phenomena.**

Persistent common requirements for scientific data management include:

- ▶ Automation of data and metadata processing;
- ▶ Parallel data processing;
- ▶ Online processing;
- ▶ Integration of multifarious data and metadata; and
- ▶ Efficient manipulation of data/metadata residing in files.

Lack of complete solutions using commercial DBMSs has led scientists in all fields to develop or adopt application-specific solutions, though some have been added on top

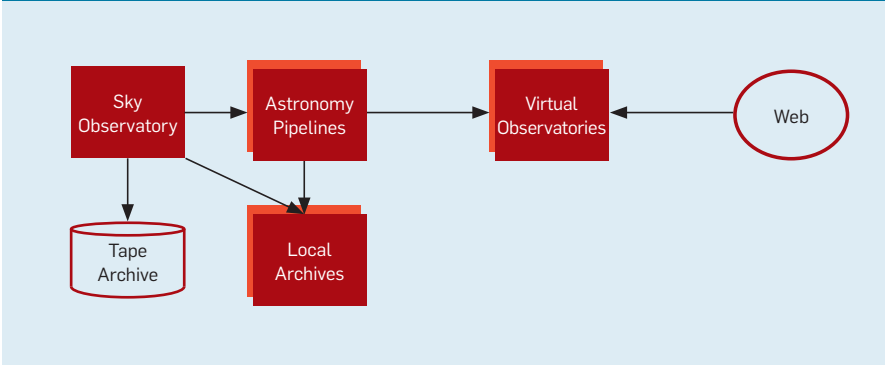
of commercial DBMSs; for example, the Sloan Digital Sky Survey (SDSS-1 and SDSS-2; <http://www.sdss.org/>) uses SQL Server as its backend. Moreover, the resulting software is typically tightly bound to the application and difficult to adapt to changes in the scientific landscape. Szalay and Blakeley<sup>9</sup> wrote, “Scientists and scientific institutions need a template and best practices that lead to balanced hardware architectures and corresponding software to deal with these volumes of data.”

Despite the challenges, the data-management research community

continues to envision a general-purpose scientific data-management system adapting current innovations: parallelism in data querying, sophisticated tools for data definition and analysis (such as clustering and SDSS-1), optimization of data organization, data caching, and replication techniques. Promising results involve automated data organization, provenance, annotation, online processing of streaming data, embedded complex data types, support for declarative data, process definition, and incorporation of files into DBMSs.

Scientific databases cover a wide

Figure 1. Workflow of SDSS data.



scope, with notable demand for high performance and data quality. Scientific data ranges from medical and biological to community science and from large-scale institutional to local laboratories. Here, we focus on the big amounts of data collected or produced by instruments archived in databases and managed by DBMSs. The database community has expertise that can be applied to solve the problems in existing scientific databases.

**Observation and Simulation**

Scientific data originates through observation and/or simulation.<sup>16</sup> Observational data is collected through detectors; input is digitized, and output is raw observational data. Simulation data is produced through simulators that take as input the values of simulation parameters. Both types of data are

often necessary for scientific research on the same topic; for instance, observational data is compared with simulation data produced under the same experimental setup. Consider three examples, one each for observational, simulation, and combined:

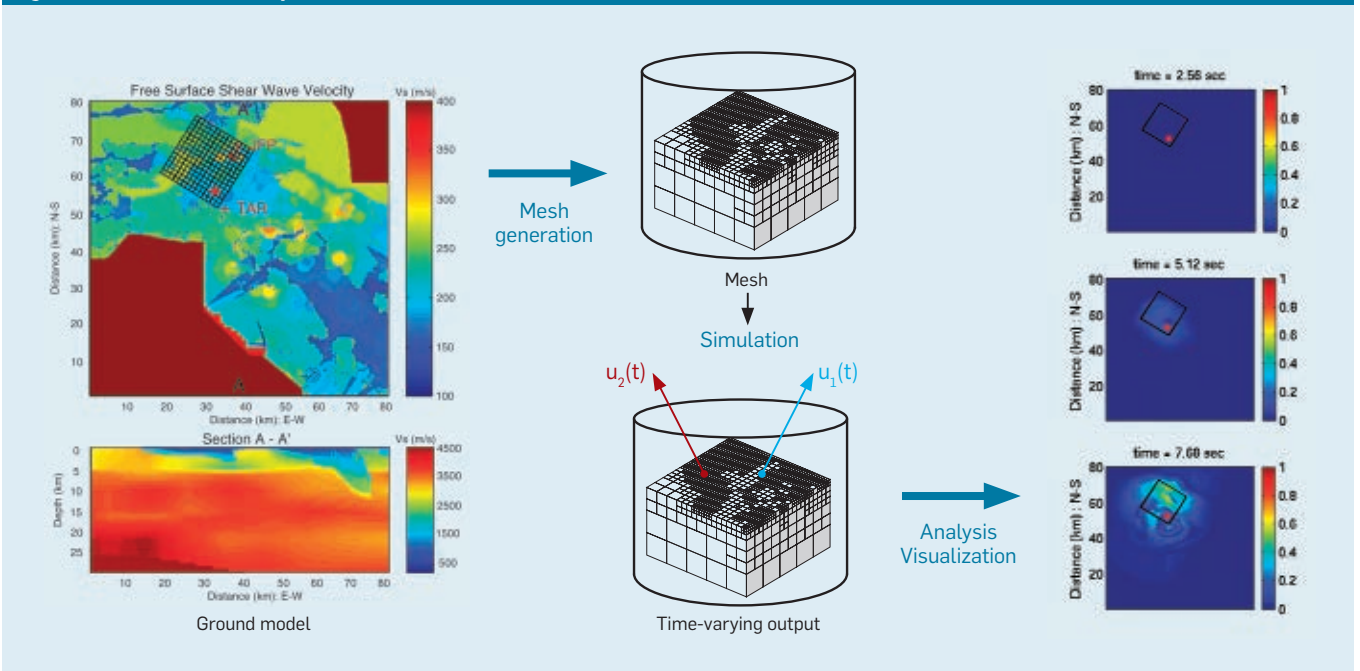
*Observational scientific data.* The SDSS located at Ohio State University and Johns Hopkins University, is a long-running astronomy project. Since 2000, it has generated a detailed 3D map of about 25% of the sky (as seen from Earth) containing millions of galaxies and quasars. One reason for its success is its use of the SQL Server DBMS. The SDSS uses the telescope at Apache Point Observatory, NM, to scan the sky at regular intervals to collect raw data. Online processing is done on the data to detect the stars and galaxies in the region. This online

processing also helps detect and fine-tune the telescope’s alignment. The image data is then recorded onto tape for archival purposes.

The tapes are physically mailed to a processing center at Fermilab in Batavia, IL, to be processed through automated software pipelines to identify celestial objects. Many astronomical-processing-software pipelines process the data in parallel. The output of this processing, along with the image data, is stored in the local archives at Fermilab. The metadata generated from the processing pipelines is converted to relational format and stored in a MS-SQL Server database. Astronomers closely associated with the SDSS access the data from the local archives (see Figure 1).

The data is also published (publicly) once every two years through virtual observatories (<http://www.sdss.org/dr7>, the final release of the SDSS-II project) by running SQL on the database at the observatories or downloading the entire database over the Internet while running the queries locally. The SDSS project began providing public data sets in 2002 with three such observatories located at the Space Telescope Science Institute in the U.S., the National Astronomical Observatory of Japan, and the Max Planck Institute for Astrophysics in Germany.

Figure 2. Workflow of earthquake-simulation data.



The schema of SDSS data includes more than 70 tables, though most user queries focus on only a few of them, referring, as needed, to spectra and images. The queries aim to spot objects with specific characteristics, similarities, and correlations. Patterns of query expression are also limited, featuring conjunctions of range and user-defined functions in both the predicate and the join clause.

*Simulation scientific data.* Earth science employs simulation models to help predict the motion of the ground during earthquakes. Ground motion is modeled with an octree-based hexahedral mesh<sup>19</sup> produced by a mesh generator, using soil density as input (see Figure 2). A “solver” tool simulates the propagation of seismic waves through the Earth by approximating the solution to the wave equation at each mesh node. During each time step, the solver computes an estimate of each node velocity in the spatial directions, writing the results to the disk. The result is a 4D spatio-temporal earthquake data set describing the ground’s velocity response. Various types of analysis can be performed on the data set, employing both time-varying and space-varying queries. For example, a user might describe a feature in the ground-mesh, and the DBMS finds the approximate location of the feature in the simula-

tion data set through multidimensional indexes.

*Combined simulation and observational data.* The ATLAS experiment (<http://atlas.ch/>), a particle-physics experiment in the Large Hadron Collider (<http://lhc.web.cern.ch/lhc/>) beneath the Swiss-French border near Geneva, is an example of scientific data processing that combines both simulated and observed data. ATLAS intends to search for new discoveries in the head-on collision of two highly energized proton beams. The entire workflow of the experiment involves petabytes of data and thousands of users from organizations the world over (see Figure 3).

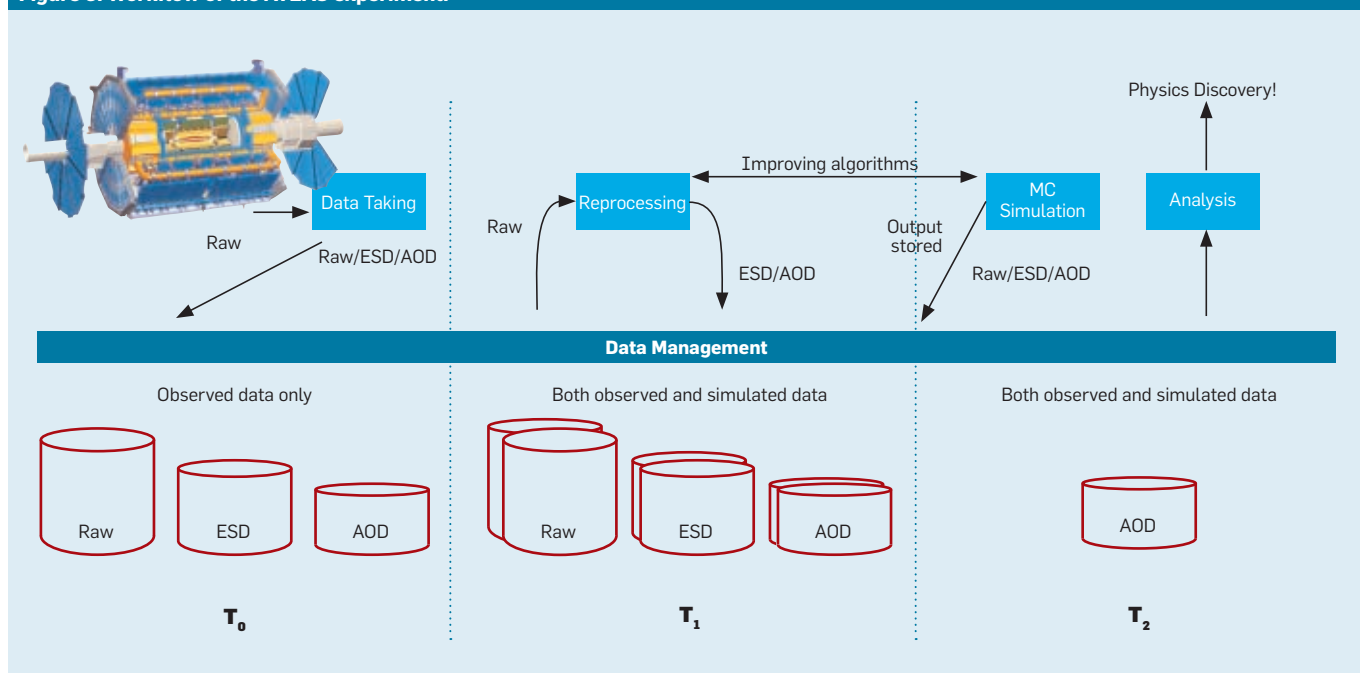
We first describe some of major ATLAS data types: The raw data is the direct observational data of the particle collisions. The detector’s output rate is about 200Hz, and raw data, or electrical signals, is generated at about 320MB/sec, then reconstructed using various algorithms to produce event summary data (ESD). ESD has an object-oriented representation of the reconstructed events (collisions), with content intended to make access to raw data unnecessary for most physics applications. ESD is further processed to create analysis object data (AOD), a reduced event representation suitable for user analysis. Data volume decreases gradually from raw to ESD

to AOD. Another important data type is tag data, or event-level metadata, stored in relational databases, designed to support efficient identification and selection of events of interest to a given analysis.

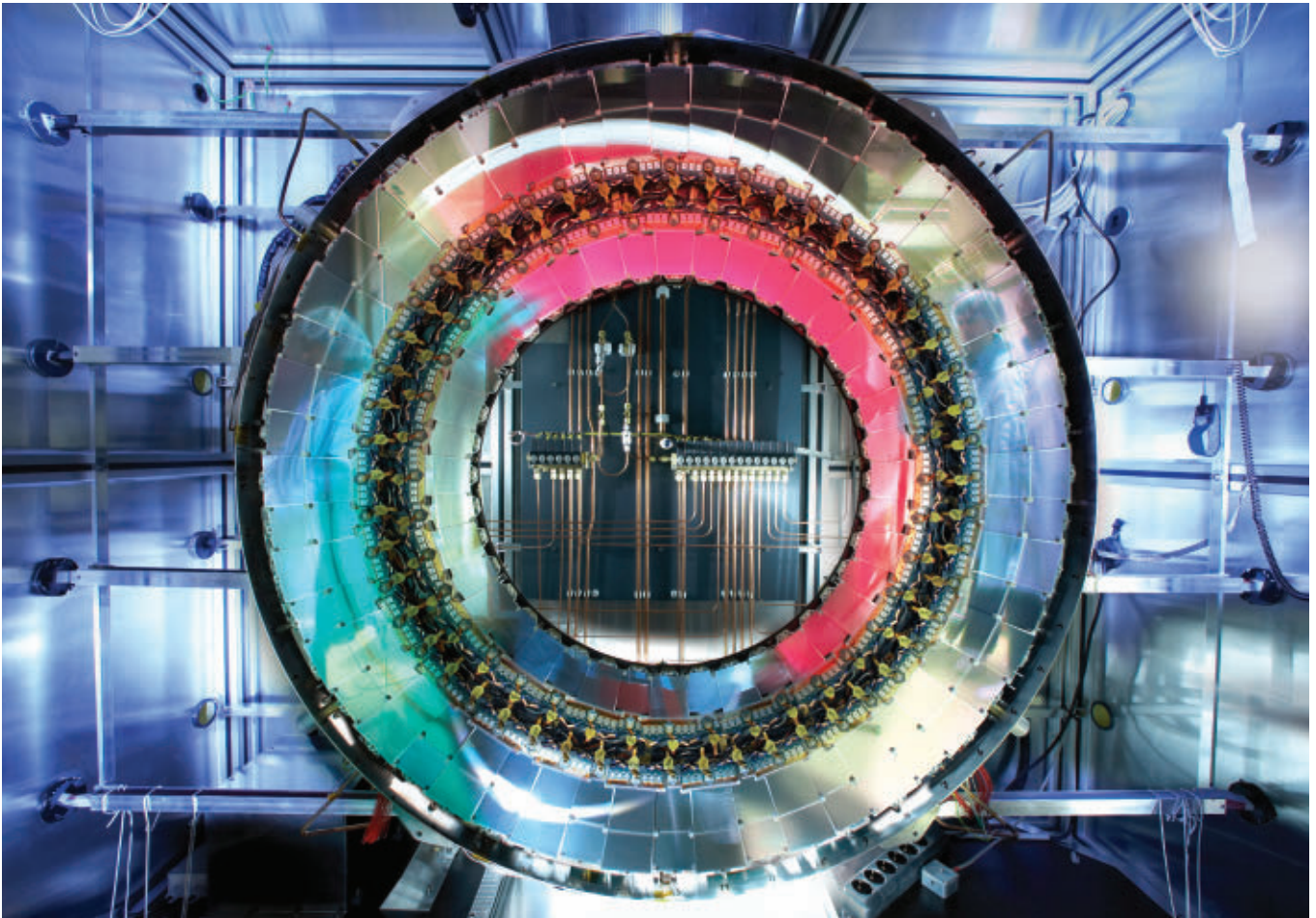
Due to the complexity of the experiment and the project’s worldwide scope, participating sites are divided into multiple layers. The Tier-0 layer is a single site—CERN itself—where the detector is located and the raw data is collected. The first reconstruction of the observed electrical signals into physics events is also done at CERN, producing ESD, AOD, and tag data. Tier-1 sites are typically large national computing centers that receive replicated data from the Tier-0 site. Tier-1 sites are also responsible for reprocessing older data, as well as for storing the final results from Monte Carlo simulations at Tier-2 sites. Tier-2 sites are mostly institutes and universities providing computing resources for Monte Carlo simulations and end-user analysis. All sites have pledged computing resources, though the vast majority is not dedicated to ATLAS or to high-energy physics experiments.

The Tier-0 site is both computation- and storage-intensive, since it stores the raw data and performs the initial event reconstruction. It also serves data to the Tier-1 sites, with aggregate sustained transfer rates for raw, ESD,

Figure 3. Workflow of the ATLAS experiment.







**Disk with silicon sensors as an endcap of the ATLAS silicon strip detector in its testbox at the Nationaal Instituut voor Subatomaire Fysica, Amsterdam, The Netherlands.**

and AOD in excess of 1GB/sec over the WAN or dedicated fiber. Tier-1 sites are also computation- and storage-intensive, since they store new data samples while permanently running reconstruction of older data samples with newer algorithms. Tier-2 sites are primarily CPU-intensive, since they generally run complex Monte Carlo simulations and user analyses while only transiently storing data, with archival copies of interesting data kept at Tier-1s sites.

The ATLAS experimental workflow involves a combination of observed and simulated data, as outlined in Figure 3. The “Data Taking” component consumes the raw data and produces ESD, AOD, and tags that are replicated to a subset of the Tier-1s sites and, in the case of AOD, to all Tier-1s and Tier-2s, where each Tier-2 site receives AOD only from its parent Tier-1 site. The “Reprocessing” component at each Tier-1 site reads older data and produces new versions of ESD and

AOD data, which is sent to a subset of other Tier-1s sites and, in the case of AOD, to all sites. The primary difference between the first reconstruction at the Tier-0 site and later reconstructions at the Tier-1 sites is due to better understanding of the detector’s behavior. Simulated data is used for this reconstruction.

Simulated data, using Monte Carlo techniques, is required to understand the behavior of the detector and help validate physics algorithms. The ATLAS machine is physically very large and complex, at 45 meters  $\times$  25 meters and weighing more than 7,000 tons, including more than 100 million electronic channels and 3,000 kilometers of cable. A precise understanding of the machine’s behavior is required to fine-tune the algorithms that process its data and reconstruct simulation data from the observed electrical signals. This is the role of the Monte Carlo simulations, using large statistics and enough data to compensate

and understand the machine’s bias. These simulations are run at Tier-2 sites by the “MC Simulation” component in Figure 3, with results sent to a Tier-1 site.

Finally, the “User Analysis” component aims to answer specific physics questions, using a model built with specific analysis algorithms. This model is then validated and improved against Monte Carlo data to compensate for the machine’s bias, including background noise, and validated against real, observed data, eventually testing the user’s hypothesis.

Beyond observational and simulation data and its hybrids, researchers discuss special cases of “information-intensive” data.<sup>16</sup> Sociology, biology, psychology, and other sciences apply research on heterogeneous data derived from both observation and simulation under various conditions. For example, in biology, research is conducted on data collected by biologists under experimental conditions

intended for a variety of purposes. Scientists involved in combustion chemistry, nanoscience, and the environment must perform research on data related to a variety of phenomena concerning objects of interest ranging from particles to devices and from living organisms to inorganic substances. Information-intensive data is characterized by heterogeneity, in both representation and the way scientists use it in the same experiment. Management of such data emphasizes logical organization and description, as well as integration.

Independent of the data type characterizing particular scientific data, its management is essentially divided into coarse phases: workflow management, management of metadata, data integration, data archiving, and data processing:

*Workflow management.* In a simple scenario, a scientific experiment is performed according to a workflow that dictates the sequence of tasks to be executed from the beginning until the end of the experiment. The tasks coarsely define the manner and means for implementing the other phases of data management: data acquisition, metadata management, data archiving, and data processing. Task deployment may be serial or parallel and sequential or looping. Broad and long-lasting experiments encompass sets and hierarchies of partial experimental studies performed according to complex workflows. In general, many experiments could share a workflow, and a workflow could use results from many experiments.

Scientific workflow management systems have been a topic for much research over the past two decades, involving modeling and enactment<sup>15</sup> and data preservation.<sup>12</sup> Numerous related scientific products are used in the sciences.

*Metadata management.* Raw data is organized in a logically meaningful way and enriched with respective metadata to support the diagnosis of unexpected (independent) reinvestigation or reinterpretation of results, ideally automatically. Metadata includes information on data acquisition (such as parameters of the detectors for observational data and simulation for simulation data), as

well as administrative data about the experiments, data model, and measurement units. Other kinds of metadata are extracted from raw data, possibly through ontologies. Metadata may also denote data relationships and quality. Annotating the data, all metadata (accumulated or extracted) is critical to deducing experimental conclusions. The metadata and workflows often complement one another, necessitating combined management.<sup>7</sup>

*Data and process integration.* Data and its respective metadata may be integrated such that they can be manipulated as a unit. Moreover, newly collected data may be integrated with historic data representing different versions or aspects of the same experiment or belonging to different experiments in the same research track. The quality of the data, as well as its semantic interpretation, is crucial for data integration. Data quality can be achieved through data cleansing; semantic integration can be achieved through ontologies. Beyond data integration, process integration is often needed to simplify the overall flow of the experiment and unify partial results. For example, integration may be necessary or desirable for data mining algorithms that facilitate feature extraction and provenance. Process integration might necessitate creation or customization of middleware allowing for interoperability among different procedures and technologies. Automatic integration of data and processes is highly desirable for ease of use but also because querying information as a unit allows parallel and online processing of partial results. Moreover, scientists want transparent access to all data. Automatic integration assumes customizable tools implementing generic integration solutions appropriate for scientific data. A notably challenging task is how to identify commonalities in scientific experiments and data in order to create integration tools.

*Data archiving.* After they've met the expected standards of data content, scientists archive the data so other scientists are able to access and use it in their own research. The data must first be stored using robust, reliable storage technology, a mandatory

task, especially for raw observational data derived from experiments that cannot be replayed or replayed only at prohibitively high cost. The complete data set is usually archived on tape, with selected parts stored on disks. Portions of the data might also have to be cached temporarily during migration between tape and disk or between various computers.

Beyond archiving master copies, data replication on multiple sites may be necessary to accommodate geographically dispersed scientists. All the challenges of distributing and replicating data management come into play when coordinating the movement of large data volumes. Efforts are under way to manage these replication tasks automatically.<sup>4</sup>

Archiving scientific data is usually performed by storing all past data versions, as well as the respective metadata (such as documentation or even human communication like email). Nevertheless, the problem of organizing archives relates to the general research problem of data versioning, so solutions to the versioning problem can be adapted to archiving scientific data. Representative versioning solutions (such as the concurrent versions system) compute differences between sequential versions and use the differences for version reconstruction. Recent proposals targeting scientific data<sup>3</sup> exploit the data's hierarchical structure in order to summarize and merge versions.

*Data processing.* Data is analyzed to extract evidence supporting scientific conclusions, ultimately yielding research progress. Toward this end, the data must undergo a series of procedures specified by scientists in light of the goal of their respective experiments. These procedures usually involve data clustering, mining, and lineage, leading to the inference of association rules and abnormalities, as well as to computation for feature identification and tracking.

Data analysis is often tightly correlated with data visualization, especially when it comes to simulation data. Scientists want a visual representation of the data to help them recognize coarse associations and abnormalities. Interleaving the steps involved in visualization and analysis yields the

right data regions for testing hypotheses and drawing conclusions. The key to efficient data processing is a carefully designed database and is why automated physical database design is the subject of recent research (discussed in the following section). In addition, there is an imminent need for online data processing (discussed in the second following section).


### Automation

Errors and inefficiencies due to human-handled physical database design are common in both metadata management and data processing. Much recent research has focused on automating procedures for these two phases of scientific data management.


*Metadata management.* Metadata processing involves determining the data model, annotations, experimental setup, and provenance. The data model can be generated automatically by finding dependencies between different attributes of data.<sup>10</sup> However, experimenters typically determine the model since this is a one-time process, and dependencies  $A=\pi r^2$  are easily identified at the attribute level.

Annotations are meta-information about the raw scientific data and especially important if the data is not numeric. For example, annotations are used in biology and astronomy image data. Given the vast scale of scientific data, automatically generating these annotations is essential. Current automated techniques for gathering annotations from documents involve machine-learning algorithms, learning the annotations through a set of pre-annotated documents.<sup>14</sup> Similar techniques are applied to images and other scientific data but must be scaled to terabyte or petabyte scale. Once annotations are built, they can be managed through a DBMS.

Experimental setups are generally recorded in notebooks, both paper and electronic, then converted to query-able digital records. The quality of such metadata is typically enforced through policies that must be as automated as possible. For example, when data is collected from instruments, instrument parameters can be recorded automatically in a database. For manually generated data, the policies must be enforced automatically.



**Not having to read from the disk and write computation results back saves hours to days of scientific work, giving scientists more time to investigate the data.**



For the ATLAS experiment, the parameters of the detectors, as well as the influence of external magnetic devices and collider configurations, are all stored automatically as metadata. Some policies can be enforced automatically through a knowledge base of logical statements; the rest can be verified through questionnaires. Many commercial tools are available for validating policies in the enterprise scenario, and the scientific community can borrow technology from them to automate the process (<http://www.compliancehome.com/>).

Provenance data includes experimental parameters and task history associated with the data. Provenance can be maintained for each data entry or for each data set. Since workload management tracks all tasks applied to the data, it can automatically tag it with task information. Hence, automating provenance is the most straightforward of the metadata-processing automation tasks. The enormous volume of automatically collected metadata easily complicates the effort to identify the relevant subset of metadata to the processing task in hand. Some research systems are capable of automatically managing a DBMS's provenance information.<sup>2</sup>

*Data processing.* Data processing depends on how data is physically organized. Commercial DBMSs usually offer a number of options for determining how to store and access it. Since scientific data might come in petabyte-scale quantities and many scientists work on the same data simultaneously, the requirements for efficient data organization and retrieval are demanding. Furthermore, the data might be distributed or replicated in multiple geographically dispersed systems; hence, network resources play an important role in facilitating data access. Possibly hundreds or thousands of scientists could simultaneously query a petabyte-scale database over the network, requiring more than 1GB/sec bandwidth.

To speed data access, the database administrator might have to tune several parameters, changing the data's logical design by normalizing the data or its physical design. The logical design is determined by the data model in the metadata-processing



phase of a science experiment. The physical design determines optimal data organization and location, caching techniques, indexes, and other performance-enhancing techniques. All depend on the data-access pattern, which is dynamic, hence, it changes much more frequently than logical design; physical design automation is therefore critical for efficient data processing.

Considering the number of parameters involved in the physical design of a scientific database, requiring the database administrator to specify and optimize the parameters for all these techniques is unreasonable. Data storage and organization must be automated.

All DBMSs today provide techniques for tuning databases. Though the provision of these techniques is a step in the right direction, existing tools are insufficient for four main reasons:

*Precision.* They require the query workload to be static and precise;

*Relational databases.* They consider only auxiliary structures to be built on relational databases and do not consider other types of data organization;

*Static database.* They assume a static database, so the statistics in the database are similar to the statistics at the time the tool is run; and

*Query optimizer.* They depend on the query optimizer to direct their search algorithms, making them slow for large workloads.

Recent database research has addressed these inherent DBMS limitations. For example, some techniques do not require prespecifying the workload,<sup>1</sup> and others make the cost model more efficient, enabling more thorough search in the data space.<sup>18</sup> However, they also fall short in several areas; for example, they are not robust enough to change database statistics and do not consider data organization other than relational data. Likewise, data-organization methods for distributed data and network caches are nascent today. Automatically utilizing multiple processing units tuned for data-intensive workloads to scale the computation is a promising research direction, and systems (such as Gray-Wulf<sup>24</sup>) apply this technique to achieve scalability.

Physical and logical design-automation tools must consider all parameters and suggest optimal organization. The tools must be robust to small variations in data and query changes, dynamically suggesting changes in the data organization when the query or data changes significantly.

### Online Processing

Most data-management techniques in the scientific community are offline today; that is, they provide the full result of the computation only after processing an entire data set. However, the ever-growing scale of scientific data volume necessitates that even simple processes, one-time data movement, checksum computation, and verification of data integrity might have to run for days before completion.

Simple errors can take hours to be noticed by scientists, and restarting the process consumes even more time. Therefore, it is important that all processing of scientific data be performed online. Converting the processes from offline to online provides the following benefits:

*Efficiency.* Many operations can be applied in a pipeline manner as data is generated or move around. The operations are performed on the data when already in memory, which is much closer to the CPU than to a disk or tape. Not having to read from the disk and write computation results back saves hours to days of scientific work, giving scientists more time to investigate the data.

*Feedback.* Giving feedback to the operations performed on the scientific data is important, because it allows scientists to plan their analysis according to the progress of the operation. Modern DBMSs typically lack a progress indicator for queries, hence scientists running queries or other processes on DBMSs are typically blind to the completion time of their queries. This blindness may lead to canceling the query and issuing a different one or abandoning the DBMS altogether. DBMSs usually allow a query issuer to compute the “cost” of a query in a unit specific to the DBMS. This cost is not very useful to scientists, since it doesn’t correspond to actual running time or account for the complete set of resources (such as memory size, band-

width, and operation sharing) available to the DBMS for running the query. Operations, including querying/ updating data, should thus provide real-time feedback about the query progress to enable scientists to better plan their experiments.

*Debugging.* Scientific data is typically processed on multiprocessor systems, as scientific applications are often parallelizable and computation can thus scale to data volume. However, it is nearly impossible to detect all the problems of a parallel program at development time. Using source debuggers for parallel programming is infeasible, since debuggers change the timing of the programs, thereby hiding many problems. Debugging becomes even more difficult when programs execute complex tasks (such as queries with user-defined functions).

Some research DBMS prototypes provide feedback on query progress,<sup>13</sup> though they are not yet incorporated into commercial systems, so the benefits are still not available to scientists. Similarly, tools that provide online visualization of progress for specific simulations are not generic enough for a variety of scientific experiments.

*Computational steering.* Building complex simulations is a challenge even in uniprocessor systems. After building them, the submitters-scientists often boot on suboptimal parameters, unaware that they’re indeed relying on suboptimal parameters until the entire simulation is over. Therefore, online processing, combined with online visualization, can simultaneously help debug such programs and parameters. The system’s operations should allow an observer to generate snapshots of the simulations or operations and, if possible, control the simulation to remove potential problems. Manual intervention in an otherwise automatic process is called “computational steering”; for example, in parallel programs, observers could decide which deadlocks to break or when simulations should change a parameter on the fly.

Software for computational steering includes the scientific programming environment SciRun (<http://www.sci.utah.edu/cibc/software/106-scirun.html>). Nevertheless, software must support simulations with pet-

abytes of data and execution of complex tasks. For software designers, it may sometimes be beneficial to model the simulations and data processing as event generators, using streaming and complex event-processing techniques to summarize data operations with little overhead or controllable accuracy guarantees.

### Data and Process Integration

Large-scale experiments organized by scientists collect and process huge amounts of raw data. Even if the original data is reorganized and filtered in a way that keeps only the interesting parts for processing, these interesting parts are still big. The reorganized data is augmented with large volumes of metadata, and the augmented reorganized data must be stored and analyzed.

Scientists must collaborate with computer engineers to develop custom solutions supporting data storage and analysis for each experiment. In spite of the effort involved in such collaborations, the experience and knowledge gained this way is not generally disseminated to the wider scientific community or benefit next-generation experimental setups. Computer engineers must therefore develop generic solutions for storage and analysis of scientific data that can be extended and customized to reduce the computing overhead of time-consuming collaborations. Developing generic solutions is feasible, since many low-level commonalities are available for representing and analyzing experimental data.

*Management of generic physical models.* Experimental data tends to have common low-level features not only across experiments of the same science, but across all sciences. For example, reorganized raw data enhanced with metadata usually involves complex structures that fit the object-oriented model. Scientific data representation benefits from inheritance and encapsulation, two fundamental innovations of the object-oriented data model.

Beyond its complexity in terms of representation, scientific data is characterized by complex interdependencies, leading to complex queries during data processing and analysis. Even

though the object-oriented model is suitable for the representation of scientific data, it cannot efficiently optimize and support complex queries.

Nevertheless, most scientific data is represented by objects with strong commonalities with respect to their structural elements. DBMSs must be extended so they manage the common structural elements of scientific data representations as generic database objects, and database support for these objects must include efficient ways to store and index data.

Experimental data derived from simulations is frequently represented as meshes ranging from structured to unstructured and consisting of tetrahedra, hexahedra, or  $n$ -facets cells. For example, an earthquake simulation data set may be represented as an unstructured hexahedral mesh. A typical volume of earth, say,  $100\text{km} \times 100\text{km} \times 30\text{km}$ , is represented by a mesh consisting of roughly one billion nodes and one billion elements requiring about 50GB of storage; such a mesh is capable of resolving seismic waves up to 2Hz. Scientific data management would benefit greatly if DBMSs offered storage and indexing methods for meshes. Initial efforts toward supporting meshes in DBMSs were presented in research<sup>19</sup> and commercial products.<sup>8</sup>

Multidimensional data also needs storage and indexing methods. Most scientific data is represented as multidimensional arrays, but support for multidimensional arrays in RDBMSs is poor. Computer engineers must produce custom solutions for manipulating multidimensional data, leading to many domain-specific data formats, including netCDF (<http://www.unidata.ucar.edu/software/netcdf/>) and HDF (<http://www.hdfgroup.org/>) for climate data; FITS (<http://heasarc.gsfc.nasa.gov/docs/heasarc/fits.html>) for astronomical data; and ROOT (<http://root.cern.ch/drupal/>) for high-energy physics data.

An experimental study<sup>5</sup> showed that, even if using array primitives in RDBMSs, native file formats outperformed the relational implementation by a factor of 20 to as much as 80. Proposed scientific DBMSs<sup>6,23</sup> provide multidimensional arrays as first-class types, aiming to bridge the gap be-

tween DBMSs and native files in the process. The multidimensional arrays are present in multidimensional online analytical processing (MOLAP) implementations from mainstream DBMSs that allow fast exploratory analysis of the data by pre-computing aggregations on multiple dimensions. However, MOLAP needs a significant amount of offline processing and an enormous amount of disk space to store the pre-computed aggregations, making them unsuitable for the enormous scale of scientific data. Attempts to support exploratory ad hoc OLAP queries on large data sets, including wavelets, promises to enable fast, powerful analysis of scientific data.<sup>21</sup>

A frequently used type of scientific data is time-and-space-based observations, meaning interesting data sets are trajectories in space and time. As trajectories are not inherently supported by databases, data points are usually stored individually and processed for line-fitting during scientific analysis. Line-fitting is a resource-consuming task and could be avoided if a DBMS inherently supported trajectories. Inherent support for trajectories is related to multidimensional array support, since trajectories are actually polylines, and each line can be approximated (fitted) by functions. Promising research results have been reported for managing trajectories.<sup>22</sup>

DBMSs should inherently support new data types while accounting for the specialized use of the new types for representing scientific data. For example, the Hierarchical Triangular Mesh method<sup>11</sup> subdivides spherical surfaces so objects localized on a sphere can be indexed and queried efficiently. Scientific data is usually persistent, meaning it is rarely (if ever) changed or involved in complex associations. While developing support mechanisms (such as indexes) for new data types, priority must go to search rather than to update efficiency. The persistence of scientific data alleviates a major requirement, making it possible to develop efficient indexes for the new types.

*Management of generic data processing.* Scientific data processing differs from experiment to experiment and discipline to discipline. No matter how wide the scope of processing

and overall heterogeneity, processing frequently encompasses generic procedures.


It is most common that scientific data is searched in order to find interesting regions with respect to prespecified characteristics, data regions, and abnormalities. Moreover, metadata processing consists of data annotation, as well as feature extraction. Data tagged with parameter values refers to the condition of the experiment and is mined to deduce common characteristics and behavior rules.

Generic processes that produce metadata (such as those just mentioned) must be supported inherently by the DBMS for all generic physical models in a parameterized manner, thus bringing processing close to the data and leading to reduced data movement and reorganization, along with efficient processing execution in the DBMS. Each generic physical model must include templates supporting the generic procedures for the model in a customizable manner. For example, the particles in the ATLAS experiment are tracked using spatio-temporal attributes. Even though the data sets are enormous, only a small amount of space and time are populated by particles. Therefore, the data sets would benefit from a generic DBMS customizable procedure supporting compression.


Scientists would benefit even more if the definition and customization of the templates could be performed using a declarative language. Such a language would give users intuitive guidance as to the specification of the customization procedure, as well as to the combination and pipelining of multiple procedures. In this way, the processing burden would be leveraged to the DBMS, and scientists would not have to function as computer engineers.

### File Management

The vast majority of scientific data is stored in files and manipulated through file systems, meaning all processing, from search to computation, is performed in the content of the files. Sophisticated frameworks have been proposed to manage the files over a large number of disks, including storage resource management



**DBMSs must be extended so they manage the common structural elements of scientific data representations as generic database objects, and database support for these objects must include efficient ways to store and index data.**



technology (<https://sdm.lbl.gov/srm-wg/index.html>).

Existing persistent scientific data in files is huge and will not be moved to databases, even if they support efficient scientific experimentation. Moreover, the tradition in applications that manipulate scientific data files is long, and implementing the same functionality in modules that are plug-able on DBMSs needs further effort. A long tradition and the need for plug-able capabilities mean that a full-fledged querying mechanism for files, similar to DBMSs, is needed. Such a querying mechanism can be constructed in either of two ways: enhance current DBMSs so they uniformly manage both structured data and unstructured data in files; and create a management layer on top of both the DBMS and the file system to enable transparent querying of structured and unstructured data. Each approach has advantages and disadvantages.

Enhancing a DBMS to manage files and data means that all mechanisms in the system should be extended for files. Querying on files is assumed to be efficient since it would benefit from sophisticated database structures (such as indexes, autonomic database organization, and database query planning). Moreover, querying structured and unstructured data is an opportunity for tight interaction among queries and query results and refined optimization in intermediate querying steps.

Extending DBMSs to manage files is a challenge for the data-management community since it entails reconsideration of many database protocols and a total rebuild of all database procedures with new enhancements for unstructured data. Yet it is inevitable that such a breakthrough in the functionality of DBMSs will involve substantial redundancy, since a big part of database management (such as transaction management) is useless in the manipulation of scientific data. Recent research efforts seeking to integrate scientific files into DBMSs include Netlobs, a netCDF cartridge for Oracle ([http://datafedwiki.wustl.edu/images/f/ff/Dews\\_poster\\_2006.ppt](http://datafedwiki.wustl.edu/images/f/ff/Dews_poster_2006.ppt)) and Barrodale Computing Services, Ltd., on Postgres (<http://www>).



barrodale.com/) but are not generic enough for other file formats (such as HDF and ROOT). Aiming for file manipulation, the Data Format Description Language Work Group project (<http://forge.gridforum.org/projects/dfdl-wg>) is developing an XML-based language for describing the metadata of files. However, these approaches do not provide complete support for DBMS features on the files. MapReduce-based techniques for processing data stored in files<sup>17,20</sup> do not replicate DBMS functionalities and are mainly used for batch processing of files.

A management level on top of the database and file system that manages structured and unstructured data separately but transparently seems more feasible in the near future. The challenge in developing this approach is twofold. First, it is necessary to construct middleware through which users define their queries in a declarative high-level manner, but the middleware must include a mechanism that transcribes queries as input for the DBMS or file system and routes it appropriately. Second, a query mechanism dedicated to the file system must be developed; the benefit of a separate file-querying mechanism is that it includes only procedures targeted at querying, thereby avoiding implications due to complicated database mechanisms—insert, delete, update—serving database operations. However, the procedures involved in the querying mechanism must be designed and implemented from scratch, an approach that precludes uniform querying of both structured and unstructured data; it also means limited uniform query optimization and limited query-execution efficiency. Future grid middleware promises to support such integration (<http://www.ogsadai.org.uk/>), though the related research is still only nascent.

## Conclusion

Scientific data management suffers from storage and processing limitations that must be overcome for scientific research to take on demanding experimentation involving data collection and processing. Future solutions promise to integrate automation, online processing, integration,

and file management, but data manipulation must still address the diversity of experimentation tasks across the sciences, the complexity of scientific data representation and processing, and the volume of collected data and metadata. Nevertheless, data-management research in all these areas suggests the inherent management problems of scientific data will indeed be addressed and solved.

## Acknowledgments

We would like to express our gratitude to Miguel Branco of CERN for contributing the dataflow of the ATLAS experiment, allowing us to demonstrate a scientific application requiring both observational and simulation data. We would also like to acknowledge the European Young Investigator Award by the European Science Foundation (<http://www.esf.org/>).

## References

1. Bruno, N. and Chaudhuri, S. Online autoadmin: Physical design tuning. In *Proceedings of the ACM International Conference on Management of Data* (Beijing, June 11–14). ACM Press, New York, 1067–1069.
2. Buneman, P., Chapman, A., and Cheney, J. Provenance management in curated databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data* (Chicago, June 27–29). ACM Press, New York, 2006, 539–550.
3. Buneman, P., Khanna, S., Tajima, K., and Tan, W. Archiving scientific data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data* (Madison, WI, June 3–6). ACM Press, New York, 2002, 1–12.
4. Chervenak, A.L., Schuler, R., Ripeanu, M., Amer, M.A., Bharathi, S., Foster, I., Iamnitchi, A., and Kesselman, C. The Globus Replica Location Service: Design and experience. *IEEE Transactions on Parallel Distributed Systems* 20, 9 (Sept. 2009), 1260–1272.
5. Cohen, S., Hurley, P., Schulz, K.W., Barth, W.L., and Benton, B. Scientific formats for object-relational database systems: A study of suitability and performance. *SIGMOD Records* 35, 2 (June 2006), 10–15.
6. Cudre-Mauroux, P., Kimura, H., Lim, K., Rogers, J., Simakov, R., Soroush, E., Velikhov, P., Wang, D.L., Balazinska, M., Becla, J., DeWitt, D., Heath, B., Maier, D., Madden, S., Patel, J., Stonebraker, M., and Zdonik, S. A demonstration of SciDB: A science-oriented DBMS. *Proceedings of VLDB Endowment* 2, 2 (Aug. 2009), 1534–1537.
7. Davidson, S.B. and Freire, J. Provenance and scientific workflows: Challenges and opportunities. In *Proceedings of the ACM SIGMOD International Conference on Management of Data* (Vancouver, B.C., June 9–12). ACM Press, New York, 1345–1350.
8. Gray, J. and Thomson, D. *Supporting Finite-Element Analysis with a Relational Database Backend, Parts i–iii*. MSR-TR-2005-49, MSR-TR-2006-21, MSR-TR-2005-151. Microsoft Research, Redmond, WA, 2005.
9. Hey, T., Tansley, S., and Tolle, K. *The Fourth Paradigm: Data-Intensive Scientific Discovery*. Microsoft, Redmond, WA, Oct. 2009.
10. Ilyas, I.F., Markl, V., Haas, P., Brown, P., and Aboulmaga, A. CORDS: Automatic discovery of correlations and soft functional dependencies. In *Proceedings of the ACM SIGMOD International Conference on Management of Data* (Paris, June 13–18). ACM Press, New York, 2004, 647–658.
11. Kunszt, P.Z., Szalay, A.S., and Thakar, A.R. The hierarchical triangular mesh. In *Proceedings of the*

- MPA/ESO/MPE Workshop (Garching, Germany, July 31–Aug. 4). Springer, Berlin, 2000, 631–637.
12. Liu, D.T., Franklin, M.J., Abdulla, G.M., Garlick, J., and Miller, M. Data-preservation in scientific workflow middleware. In *Proceedings of the 18th International Conference on Scientific and Statistical Database Management* (July 3–5). IEEE Computer Society, Washington, DC, 2006, 49–58.
13. Mishra, C. and Koudas, N. A lightweight online framework for query progress indicators. In *Proceedings of the IEEE International Conference on Data Engineering* (Istanbul, Apr. 15–20). IEEE Press, 1292–1296.
14. Müller, A. and Sternberg, M. Structural annotation of the human genome. In *Proceedings of the German Conference on Bioinformatics* (Braunschweig, Germany, Oct. 7–10). German Research Center for Biotechnology, Braunschweig, 2001, 211–212.
15. Ngu, A.H., Bowers, S., Haasch, N., McPhillips, T., and Critchlow, T. Flexible scientific workflow modeling using frames, templates, and dynamic embedding. In *Proceedings of the 20th International Conference on Data Engineering and Statistical Database Management* (Hong Kong, July 9–11). Springer-Verlag, Berlin, Heidelberg, 2008, 566–572.
16. Office of Data Management Challenge. *Report from the DOE Office of Science Data Management Workshops*, Mar.–May 2004; <http://www.er.doe.gov/ascr/ProgramDocuments/Docs/Final-report-v26.pdf>
17. Olston, C., Reed, B., Srivastava, U., Kumar, R., and Tomkins, A. Pig Latin: A not-so-foreign language for data processing. In *Proceedings of the ACM SIGMOD International Conference on Management of Data* (Vancouver, B.C., June 9–12). ACM Press, New York, 2008, 1099–1110.
18. Papadomanolakis, S., Dash, D., and Ailamaki, A. Efficient use of the query optimizer for automated physical design. In *Proceedings of the 33rd International Conference on Very Large Data Bases* (Vienna, Austria, Sept. 23–27). VLDB Endowment, 2007, 1093–1104.
19. Papadomanolakis, S., Ailamaki, A., Lopez, J.C., Tu, T., O'Hallaron, D.R., and Heber, G. Efficient query processing on unstructured tetrahedral meshes. In *Proceedings of the ACM SIGMOD International Conference on Management of Data* (Chicago, June 27–29). ACM Press, New York, 2006, 551–562.
20. Pike, R., Dorward, S., Griesemer, R., and Quintan, S. Interpreting the data: Parallel analysis with Sawzall. *Scientific Programming* 13, 4 (Oct. 2005), 277–298.
21. Shahabi, C., Jahangiri, M., and Banaei-Kashani, F. ProDA: An end-to-end wavelet-based OLAP system for massive datasets. *Computer* 41, 4 (Apr. 2008), 69–77.
22. Spaccapietra, S., Parent, C., Damiani, M.L., de Macedo, J. A., Porto, F., and Vangenot, C. A conceptual view on trajectories. *Data Knowledge Engineering* 65, 1 (Apr. 2008), 126–146.
23. Stonebraker, M., Bear, C., Çetintemel, U., Cherniack, M., Ge, T., Hachem, N., Harizopoulos, S., Lifter, J., Rogers, J., and Zdonik, S.B. One size fits all? Part 2: Benchmarking studies. In *Proceedings of the Conference on Innovative Data Systems Research* (Asilomar, Jan. 7–10, 2007), 173–184.
24. Szalay, A., Bell, G., VandenBerg, J., et al. GrayWulf: Scalable clustered architecture for data-intensive computing. In *Proceedings of the Hawaii International Conference on System Sciences* (Waikoloa, Jan. 5–8). IEEE Computer Society Press, 2009, 1–10.

**Anastasia Ailamaki** (natassa@epfl.ch) is director of the Data-Intensive Applications and Systems Laboratory and a professor at Ecole Polytechnique Fédérale de Lausanne, Lausanne, Switzerland, and an adjunct professor at Carnegie Mellon University, Pittsburgh, PA.

**Verena Kantere** (verena.kantere@epfl.ch) is a postdoctoral researcher in the Data-Intensive Applications and Systems Laboratory at Ecole Polytechnique Fédérale de Lausanne, Lausanne, Switzerland.

**Debabrata Dash** (debabrata.dash@epfl.ch) is Ph.D student in the Data-Intensive Applications and Systems Laboratory at Ecole Polytechnique Fédérale de Lausanne, Lausanne, Switzerland.

## Conference acceptance rate signals future impact of published conference papers.

BY JILIN CHEN AND JOSEPH A. KONSTAN

# Conference Paper Selectivity and Impact

STUDYING THE METADATA of the ACM Digital Library (<http://www.acm.org/dl>), we found that papers in low-acceptance-rate conferences have higher impact than those in high-acceptance-rate conferences within ACM, where impact is measured by the number of citations received. We also found that highly selective conferences—those that accept 30% or less of submissions—are cited at a rate comparable to or greater than ACM Transactions and journals.

In addition, the higher impact of selective conferences cannot be explained solely by a more strict filtering process; selectivity signals authors and/or readers of the quality of a venue and thus invites higher-quality submissions from authors and/or more citations from other authors.

Low-acceptance-rate conferences with selective peer-review processes distinguish computer science from other academic fields where only journal publication carries real weight. Focus on conferences challenges the

### » key insights


- **Papers published in highly selective CS conferences are cited more often on average than papers published in Transactions and journals.**
- **Conference selectivity serves two purposes: pick the best submitted papers and signal prospective authors and readers about conference quality.**
- **Below a certain acceptance rate, selectivity can backfire; conferences rejecting 85% or more of their submissions risk discouraging overall submissions and inadvertently filtering out high-impact research.**

field in two ways: how to assess the importance of conference publication, particularly compared to journal publication, and how to manage conferences to maximize the impact of the papers they publish. “Impact factor” (average citation rate) is the commonly used measure of the influence of a journal on its field. While nearly all computer scientists have strong intuition about the link between conference acceptance rate and a paper’s impact, we are aware of no systematic studies examining that link or comparing conference and journal papers in terms of impact.


This article addresses three main questions: How does a conference’s acceptance rate correlate with the impact of its papers? How much impact do conference papers have compared to journal papers? To what extent does the impact of a highly selective conference derive from filtering (the selectivity of the review process) vs. signaling (the message the conference sends to both authors and readers by being selective)? Our results offer guidance to conference organizers, since acceptance rate is one of the few parameters they can control to maximize the impact of their conferences. In addition, our results inform the process of evaluating researchers, since we know that computer scientists often defend the primary publication of results in conferences, particularly when being evaluated by those outside the field (such as in tenure evaluations).<sup>2</sup> Finally, we hope these results will help guide individual researchers in understanding the expected impact of publishing their papers in the various venues.

### Data and Methodology

We based our study on ACM Digital Library metadata for all ACM conference and journal papers as of May 2007, as well as on selected other papers in the *ACM Guide to Computing Literature* for which metadata was available. Since there is no established metric for measuring the scientific influence of published papers, we chose to estimate a paper’s influence as the number of times it was cited in the two years following publication, referred to as citation count or simply as impact. We excluded from this count “self-citation” in subsequent papers by the authors



**Overall, the conference papers had an average two-year citation count of 2.15, and the journal papers had an average two-year citation count of 1.53.**



of the original work. Using citations as a measure of scientific influence has a long tradition, including the journal-impact factor.<sup>1</sup> We chose two years as a compromise between measuring long-term impact and the practical importance of measuring impact of more recent work.<sup>1</sup> Less than two years might be too short for the field to recognize the worth of a paper and cite it. More than two years would have excluded more recently published papers from our analysis due to insufficient time after publication, so would not have allowed us to include the current era of widespread use of the Digital Library.<sup>a</sup>

For conferences, we counted only full papers, since they represent their attempt to publish high-impact work, rather than posters and other less-rigorously reviewed material that might also appear in conference proceedings. Conferences where acceptance rates were not available were excluded as well. For journals, we included only titled ACM Transactions and journals; only these categories are generally viewed as archival research venues of lasting value.

Finally, since our data source was limited to the metadata in the *ACM Guide*, our analysis considered only citations from within that collection and ignored all citations from conferences and journals outside of it; this was a pragmatic constraint because, in part, other indexing services do not comprehensively index conference proceedings. While it means that all our numbers were underestimates and that the nature of the underestimates varied by field (we expected to significantly underestimate artificial intelligence and numerical-computation papers due to the large number of papers published by SIAM and AAAI outside our collection), such underestimates were not biased toward any particular acceptance rate in our data set.<sup>b</sup>

a To ensure that the two-year citation count was reasonable, we repeated this analysis using four- and eight-year citation counts; the distributions and graphs were similar, and the conclusions were unchanged.

b We hand-checked 50 randomly selected conference papers receiving at least one citation in our data set, comparing citation count in the data set against citation count according to Google scholar (<http://scholar.google.com/>). When trying to predict Google scholar citation count from ACM citation count in a linear regression, we found an adjusted *R*-square of



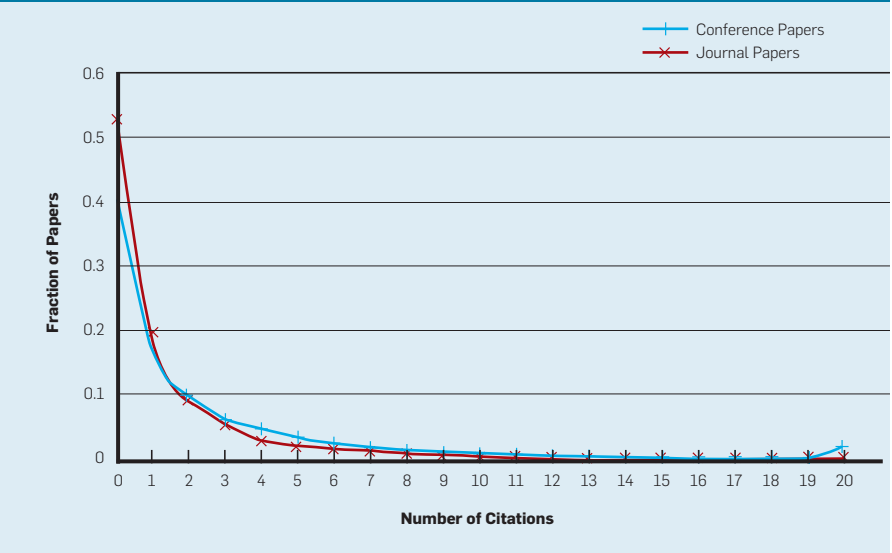
Therefore, this limitation did not invalidate our results.

Our analysis included 600 conferences consisting of 14,017 full papers and 1,508 issues of journals consisting of 10,277 articles published from 1970 to 2005. Their citation counts were based on our full data set consisting of 4,119,899 listed references from 790,726 paper records, of which 1,536,923 references were resolved within the data set itself and can be used toward citation count. Overall, the conference papers had an average two-year citation count of 2.15 and the journal papers an average two-year citation count of 1.53. These counts follow a highly skewed distribution (see Figure 1), with over 70% of papers receiving no more than two citations. Note that while the average two-year citation count for conferences was higher than journals, the average four-year citation count for articles published before 2003 was 3.16 for conferences vs. 4.03 for journals; that is, on average, journals come out a little ahead of conference proceedings over the longer term.

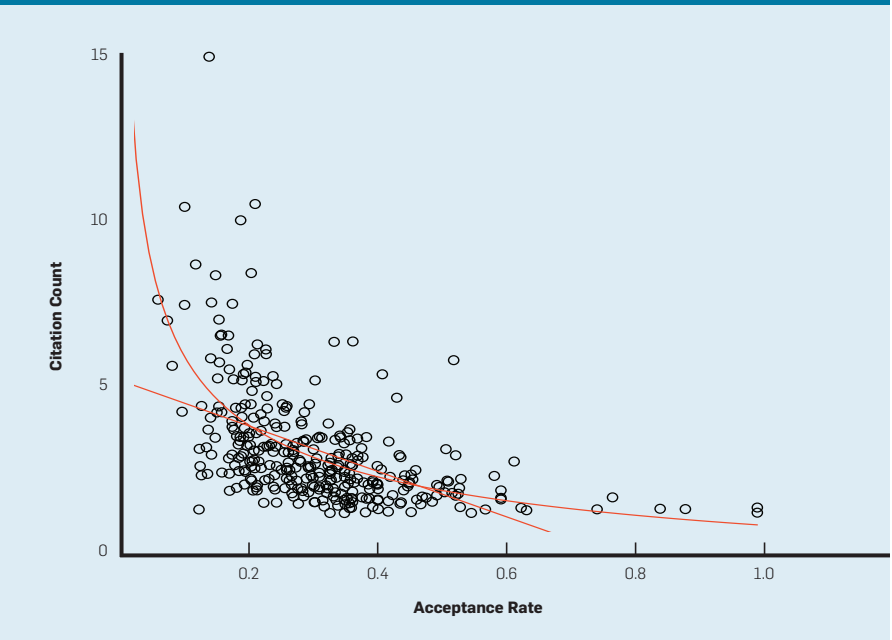
**Results**

We addressed the first question—on how a conference’s acceptance rate correlates with the impact of its papers—by correlating citation count with acceptance rate; Figure 2 shows a scatterplot of average citation counts of ACM conferences (y-axis) by their acceptance rates (x-axis). Citation count differs substantially from the spectrum of acceptance rates, with a clear trend toward more citations for low acceptance rates; we observed a statistically significant correlation between the two values (each paper treated as a sample,  $F[1, 14015] = 970.5, p < .001^c$ )

**Figure 1. Citation count distribution within two years of publication.**



**Figure 2. Average citation count vs. acceptance rate for ACM conferences.**



and computed both a linear regression line (each conference weighted by its size, adjusted  $R$ -square: 0.258, weighted residual sum-of-squares: 35311) and a nonlinear regression curve in the form of  $y = a + bx^{-c}$  (each conference weighted by its size, pseudo  $R$ -square: 0.325, weighted residual sum-of-squares: 32222), as shown in Figure 2.

Figure 3 is an aggregate view of the data, where we grouped conferences

into bins according to acceptance rates and computed the average citation counts of each bin.<sup>d</sup> Citation counts for journal articles are shown as a dashed line for comparison. Conferences with rates less than 20% enjoyed an average citation count as high as 3.5. Less-selective conferences yielded fewer citations per paper, with the least-selective conferences (>55% acceptance rate) averaging less than 1/2 citation per paper.

0.852, showing that overall ACM citation count is proportional to Google scholar citation count with a small variation. When added as an additional parameter to the regression, acceptance rate had a nonsignificant coefficient, showing that acceptance rate does not have a significant effect on the difference between ACM citation count and Google scholar citation count. We also hand-checked 50 randomly selected conference papers receiving no citations in our data set, finding no correlation between acceptance rate and Google scholar citation count.

c This F-statistic shows how well a linear relationship between acceptance rate and citation count explains the variance within citation count. The notation  $F[1, 14015] = 970.5, p < .001$  signifies one degree of freedom for model (from using only acceptance rate to ex-

plain citation counts), 14,015 degrees of freedom for error (from the more than 14,000 conference papers in our analysis), an F-statistic of 970.5, and probability less than 0.001 that the correlation between acceptance rate and citation count is the result of random chance.

d We excluded conferences with an acceptance rate less than 10% and an acceptance rate over 60%, as there were too few conferences in these categories for meaningful analysis.

Figure 4 shows the percentages of papers within each group where citation count was above a certain threshold. The bottom bands (reflecting papers cited more than 10, 15, or 20 times in the following two years) show high-acceptance-rate conferences have few papers with high impact. Also notable is the fact that about 75% of papers published in >55%-acceptance-rate conferences were not cited at all in the following two years.

Addressing the second question—on how much impact conference papers have compared to journal papers—in Figures 3 and 4, we found that overall, journals did not outperform conferences in terms of citation count; they were, in fact, similar to conferences with acceptance rates around 30%, far behind conferences with acceptance rates below 25% ( $T$ -test,  $T[7603] = 24.8$ ,  $p < .001$ ). Similarly, journals published as many papers receiving no cita-

tions in the next two years as conferences accepting 35%–40% of submissions, a much higher low-impact percentage than for highly selective conferences.

The same analyses over four- and eight-year periods yielded results consistent with the two-year period; journal papers received significantly fewer citations than conferences where the acceptance rate was below 25%.

Low-acceptance-rate conferences in computer science have a greater impact than the average ACM journal. The fact that some journal papers are expanded versions of already-published (and cited) conference papers is a confounding factor here. We do not have data that effectively tracks research contributions through multiple publications to assess the cumulative impact of ideas published more than once.

Pondering why citation count correlates with acceptance rate brings us to

the third question—on the extent the impact of a highly selective conference derives from filtering vs. signaling—as this correlation can be attributed to two mechanisms:

*Filtering.* A selective review process filters out low-quality papers from the submission pool, lowering the acceptance rate and increasing the average impact of published papers; and

*Signaling.* A low acceptance rate signals high quality, thus attracting better submissions and more future citations, because researchers simply prefer submitting papers to reading the proceedings of and citing publications from better conferences. While filtering is commonly viewed as the whole point of a review process and thus likely explains the correlation to some extent, it is unclear whether signaling is also a factor. As a result, to address the third question, we clarified the existence of signaling by separating its potential effect from filtering.

We performed this separation by normalizing the selectivity of filtering to the same level for different conferences. For example, for a conference accepting 90 papers at a 30% acceptance rate, the best potential average citation count the conference could have achieved by lowering the acceptance rate to, say, 10% for the same submission pool would be the average citation count of the top 30 most-cited papers of the 90 accepted (presumably the 30 best papers of the original 300 submitted). We treated these 30 papers as the top 10% best submissions in the pool; other submissions were either filtered out during the actual review or later received fewer citations. Their citation count was thus an upperbound estimate of what might be achieved through stricter filtering, assuming conference program committees were able to pick exactly the submissions that would ultimately be the most highly cited. Using this normalization, we compared the same top portions of submission pools of all conferences and evaluated the effect of signaling without the influence of filtering. We normalized all ACM conferences in Figure 3 to a 10% acceptance rate and compared the citation counts of their top 10% best submissions; Figure 5 (same format as Figure 3) outlines the results. We excluded transactions and journals, as we were unable to get actual

Figure 3. Average citation count by acceptance rate within two years of publication.

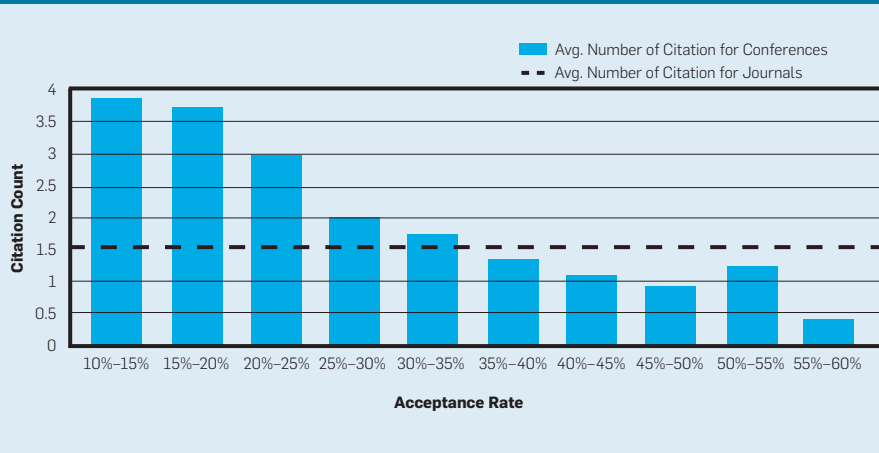
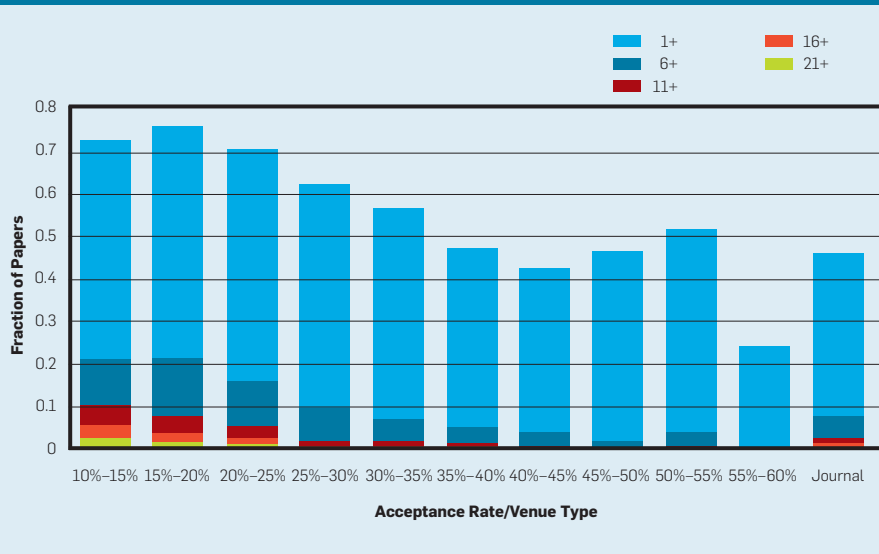


Figure 4. Citation count distribution by acceptance rate within two years of publication.



acceptance-rate data, which might also be less meaningful, given the multi-review cycle common in journals.

Figure 5 suggests that citation count for the top 10% of submitted papers follows a trend similar to that of the full proceedings ( $F[1, 5165] = 149.5, p < .001$ ), with generally higher count for low acceptance rates. This correlation indicates that filtering alone does not fully explain the correlation between citation count and acceptance rate; other factors (such as signaling) play a role.

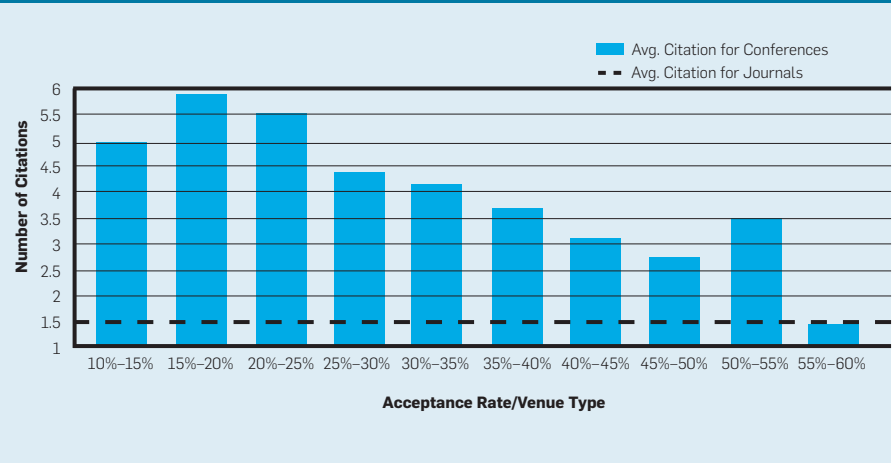
### Discussion

Combining the results in Figures 3 and 5 provides further insight into the relationship between acceptance rate and citation count. For conferences with acceptance rates over 20%, the citation numbers in the figures almost consistently drop as the acceptance rate increases, suggesting that in this range, a higher acceptance rate makes conferences lose out on citation count not only for the conference but for its best submitted papers. Either higher-quality papers are not submitted to higher-acceptance-rate conferences as frequently or those submitted are not cited because readers do not explore the conferences as often as they explore lower-acceptance-rate conferences to find them.

The case for conferences with acceptance rates below 20% is more intriguing. Note that the lower impact of the 10%–15% group compared with the 15%–20% group in Figure 5 is statistically significant ( $T[1198] = 3.21, p < .002$ ). That is, the top-cited papers from 15%–20%-acceptance-rate conferences are cited *more* often than those from 10%–15% conferences. We hypothesize that an extremely selective but imperfect (as review processes always are) review process has filtered-out submissions that would deliver impact if published. This hypothesis matches the common speculation, including from former ACM President David Patterson, that highly selective conferences too often choose incremental work at the expense of innovative breakthrough work.<sup>3</sup>

Alternatively, extremely low acceptance rates might discourage submissions by authors who dislike and avoid competition or the perception of there being a “lottery” among good papers for

**Figure 5. Average citation count vs. acceptance rate within two years of publication, top 10% of submissions.**



a few coveted publication slots. A third explanation suggests that extremely low acceptance rates have caused a conference proceedings to be of such limited focus that other researchers stop checking it regularly and thus never cite it. We consider all three to be plausible explanations; intuitively, all would hurt the impact of lower-acceptance-rate conferences more than they would hurt higher-acceptance-rate conferences.

### Conclusion

Our results have several implications: First and foremost, computing researchers are right to view conferences as an important archival venue and use acceptance rate as an indicator of future impact. Papers in highly selective conferences—acceptance rates of 30% or less—should continue to be treated as first-class research contributions with impact comparable to, or better than, journal papers.

Second, we hope to bring to the attention of conference organizers and program committees the insight that conference selectivity does have a signaling value beyond simply separating good work from bad. Adopting the right selectivity level helps attract better submissions and more citations. Acceptance rates of 15%–20% seem optimal for generating the highest number of future citations for both the proceedings as a whole and the top papers submitted, though we caution that this guideline is based on ACM-wide data, and individual conferences should consider their goals and the norms of their sub-disciplines in setting target acceptance rates. Furthermore, many conferences

have goals separate from generating citations, and many high-acceptance-rate conferences might do a better job getting feedback to early ideas, supporting networking among attendees, and bringing together different specialties.

Given the link between acceptance rate and future impact, further research is warranted in the degree to which a conference’s reputation interacts over time with changes in its acceptance rate. Though a number of highly selective conferences have become more or less selective over time, we still lack enough data to clarify the effect of such changes. We hope that understanding them will yield new insight for conference organizers tuning their selectivity in the future.

### Acknowledgments

This work was supported by National Science Foundation grant IIS-0534939. We thank our colleague John Riedl of the University of Minnesota for his valuable insights and suggestions.

### References

1. Garfield, E. Citation analysis as a tool in journal evaluation. *Science* 178, 60 (Nov. 1972), 471–479.
2. National Research Council. *Academic Careers for Experimental Computer Scientists and Engineers*. U.S. National Academy of Sciences Report, Washington, D.C., 1994; [http://www.nap.edu/catalog.php?record\\_id=2236](http://www.nap.edu/catalog.php?record_id=2236)
3. Patterson, D.A. The health of research conferences and the dearth of big idea papers. *Commun. ACM* 47, 12 (Dec. 2004), 23–24.

**Jilin Chen** (jilin@cs.umn.edu) is a doctoral student in the Department of Computer Science and Engineering at the University of Minnesota, Twin Cities.

**Joseph A. Konstan** (konstan@cs.umn.edu) is Distinguished McKnight Professor and Distinguished University Teaching Professor in the Department of Computer Science and Engineering at the University of Minnesota, Twin Cities.



**New algorithms provide the ability for robust but scalable image search.**

BY KRISTEN GRAUMAN

## Efficiently Searching for Similar Images

IF A TREE falls in the forest and no one is there to hear it, does it make a sound? In the realm of content-based image retrieval, the question is: if an image is captured and recorded but no one is there to annotate it, *does it ever again make an appearance?* Over the last decade we have witnessed an explosion in the number and throughput of imaging devices. At the same time, advances in computer hardware and communications have made it increasingly possible to capture, store, and transmit image data at a low cost. Billions of images and videos are hosted publicly on the Web; cameras embedded in mobile devices are commonplace. Climatologists compile large volumes of satellite imagery in search of long-term trends that might elucidate glacial activity and its impact on water supplies. Centralized medical image databases archive terabytes of X-ray, CAT scans, and ultrasound images, which may assist in new diagnoses.

Image and video data are certainly rich with meaning, memories, or entertainment, and in some cases they can facilitate communication or scientific discovery. However, without efficient vision algorithms to automatically analyze and index visual data, their full value will remain latent—the ratio of data to human attention is simply too large.

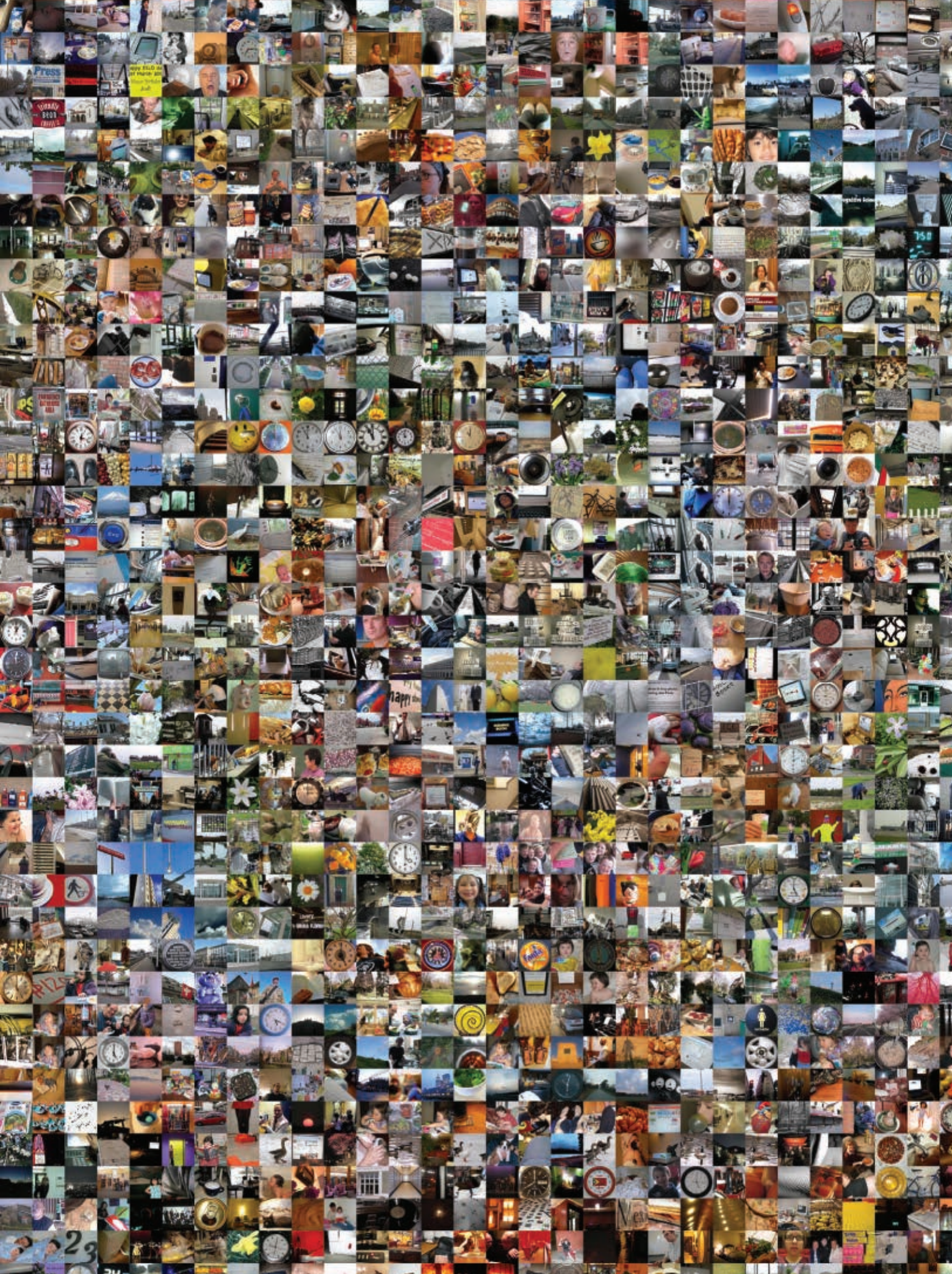
Most image search tools in operation today rely heavily on keyword meta-tags, where an image or video is annotated with a limited number of words that are either provided manually, or else are taken from whatever text occurs nearby in its containing document. While such a scheme simplifies the image indexing task to one that well-known information retrieval techniques can handle, it has serious shortcomings. At the surface, accurate manual tags are clearly too expensive to obtain on a large scale, and keywords in proximity to an image are often irrelevant.

Even more problematic, however, is the semantic disconnect between words and visual content: a word is a human construct with a precise intent, whereas a natural image can convey a multitude of concepts within its (say) million pixels, and any one may be more or less significant depending on the context. For example, imagine querying a database for all text documents containing the word “forest.” Now imagine

### » key insights

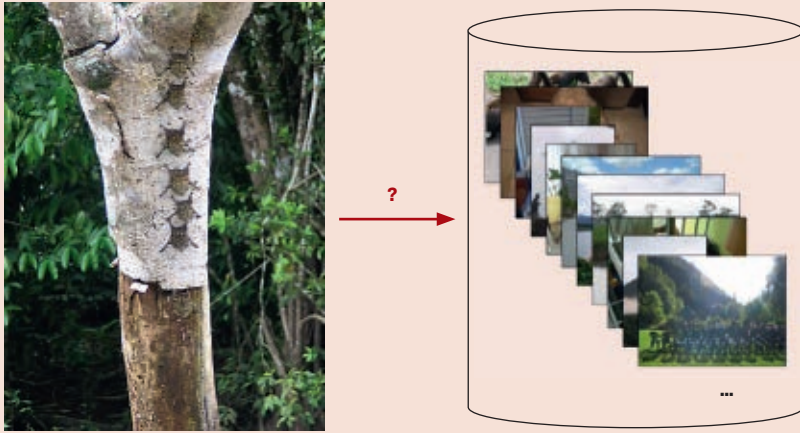
- **As it becomes increasingly viable to capture, store, and share large amounts of image and video data, automatic image analysis is crucial to managing visual information.**
- **Often the most effective metrics for image comparisons do not mesh well with existing efficient search methods; scalable image recognition and search techniques, however, aim to provide direct access to visual content.**
- **The capability to perform fast visual search is critical for many applications, and more generally, lays the groundwork for data-driven approaches in computer vision.**







**Figure 1. Visual data is complex and often holds valuable information. Image-based search algorithms automatically analyze and organize visual content, with the goal of allowing efficient retrieval from large image or video collections.**



conjuring a text query that would find you all images relevant to the one on the left in Figure 1; while you immediately have a visual concept, it may be difficult to pinpoint words to capture it, especially if the objects within the image are unfamiliar. Thus, even if we were to somehow record keywords for all the images in the world, visual data would still not be sufficiently accessible.

Content-based image search streamlines the process by sorting images directly based on their visual information and allowing images themselves to serve as queries. While early work in the area focused on correlating low-level cues such as color and texture,<sup>8,28</sup> more recently the image search problem has become intertwined with the fundamental problem of recognition, in which algorithms must capture higher-level notions of visual object and scene categories.

The technical challenges are considerable. Instances of the same object category can generate drastically different images, depending on confounding variables such as illumination conditions, object pose, camera viewpoint, partial occlusions, and unrelated background “clutter” (see Figure 2). In general, the quality of image search relies significantly on the chosen image representation and the distance metric used to compare examples. Meanwhile, the complexity of useful image representations combined with the sheer magnitude of the search task immediately raises the practical issue of scalability.

This article overviews our work considering how to construct robust measures of image similarity that can be deployed efficiently, even for complex feature spaces and massive image databases. We pose three essential technical questions: (1) what is an effective distance measure between images that can withstand the naturally occurring variability among related examples? (2) when external cues *beyond* observable image content are available, how can that improve our comparisons? and (3) what kind of search strategy will support fast queries with such image-driven metrics, particularly when our database is so large that a linear scan is infeasible? The following sections address each of these issues in turn, and highlight

**Figure 2. The same object type can generate dramatically different images due to a variety of nuisance parameters (top), but local descriptions can offer substantial robustness (bottom).**





some of our results to demonstrate the impact with real image data.

Our approach enables rapid, scalable search for meaningful metrics that were previously restricted to artificially modestly sized inputs or databases. Additionally, we show how minimal annotations can be exploited to learn how to compare images more reliably. Both contributions support the ultimate goal of harnessing the potential of very large repositories and providing direct access to visual content.

### Comparing Images with Local Feature Matches

Earlier work in content-based image retrieval focused on global representations that describe each image with a *single* vector of attributes, such as a color histogram, or an ordered list of intensity values or filter responses. While vector representations permit the direct application of standard distance functions and indexing structures, they are known to be prohibitively sensitive to realistic image conditions. For example, consider stacking the images in Figure 2 one on top of the other, and then checking the intensity at any given pixel for each example—it is quite likely that few of them would be in agreement, even though each image contains a koala as its most prominent object.

Much recent work shows that decomposing an image into its component parts (or so-called “local features”) grants resilience to image transformations and variations in object appearance.<sup>23,30</sup> Typically, one either takes a dense sample of regions at multiple scales, or else uses an interest operator to identify the most salient regions in an image. Possible salient points include pixels marking high contrast (edges), or points selected for a region’s repeatability at multiple scales (see Tuytelaars<sup>30</sup> for a survey). Then, for each detected region, a feature descriptor vector is formed. Descriptors may be lists of pixel values within a patch, or histograms of oriented contrast within the regions,<sup>23</sup> for example. The result is one set of local appearance or shape description vectors per image, often numbering on the order of 2,000 or more features per image.

The idea behind such representations is to detect strong similarity

between local portions of related images, even when the images appear quite different at the global level. Local features are more reliable for several reasons:

- **Isolate occlusions:** An object may be partially occluded by another object. A global representation will suffer proportionally, but for local representations, any parts that are still visible will have their descriptions remain intact.

- **Isolate clutter and the background:** Similarly, while the global description may be overwhelmed by large amounts of background or clutter, small parts of an image containing an actual object of interest can emerge if we describe them independently by regions. Recognition can proceed without prior segmentation.

- **Accommodate partial appearance variation:** When instances of a category can vary widely in some aspects of their appearance, their commonality may be best captured by a part-wise description that includes the shared reoccurring pieces of the object class.

- **Invariant local descriptors:** Researchers have developed local descriptors designed explicitly to offer invariance to common transformations, such as illumination changes, rotations, translations, scaling, or all affine transformations.

This appealing representation—a set of vectors—does not fit the mold of many traditional distances and learning algorithms. Conventional methods assume vector inputs, but with local representations, each image produces a variable number of features, and there is no ordering among features in a single set. In this situation, computing a correspondence or matching between two images’ features can reveal their overall resemblance: if many parts in image A can be associated with similar-looking parts in image B, then they are likely to display similar content (see Figure 2, bottom).

Current strategies for recognition and image matching exploit this notion in some form, often by building spatial constellations of a category’s reoccurring local features, summarizing images with a histogram of discretized local patches, or explicitly computing the least-cost correspondences (for a survey, see Pinz<sup>27</sup> and references

therein). However, a real practical challenge is the computational cost of evaluating the optimal matching, which is cubic in the number of features extracted per image. Compounding that cost is substantial empirical evidence showing that recognition accuracy improves when larger and denser feature sets are used.

### The Pyramid Match Algorithm

To address this challenge, we developed the *pyramid match*—a linear-time matching function over unordered feature sets—and showed how it allows local features to be used efficiently within the context of multiple image search and learning problems.<sup>12</sup> The pyramid match approximates the similarity measured by the *optimal partial matching* between feature sets of variable cardinalities. Because the matching is partial, some features may be ignored without penalty to the overall set similarity. This tolerance makes the measure robust in situations where superfluous or “outlier” features may appear. Note that our work focuses on the image matching and indexing aspects of the problem, and is flexible to the representation choice, that is, which particular image feature detectors and descriptors are used as input.

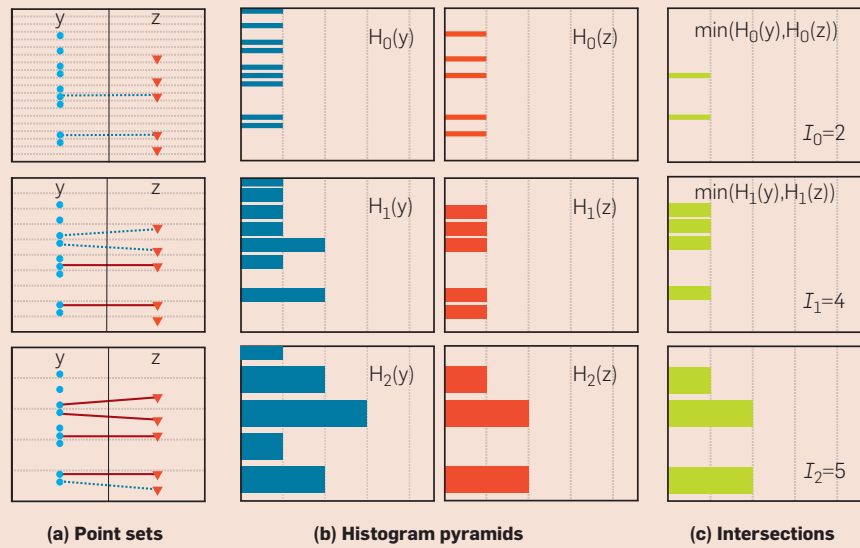
We consider a feature space  $\mathcal{V}$  of  $d$ -dimensional vectors for which the values have a maximal range  $D$ . The point sets we match will come from the input space  $S$ , which contains sets of feature vectors drawn from  $\mathcal{V}$ :  $S = \{\mathbf{X} | \mathbf{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_m\}\}$ , where each feature  $\mathbf{x}_i \in \mathcal{V} \subseteq \mathbb{R}^d$ , and  $m = |\mathbf{X}|$ . We can think of each  $\mathbf{x}_i$  as a descriptor for one of the elliptical image regions on the koalas in Figure 2. Note that the point dimension  $d$  is fixed for all features in  $\mathcal{V}$ , but the value of  $m$  may vary across instances in  $S$ .

Given point sets  $\mathbf{X}, \mathbf{Y} \in S$ , with  $|\mathbf{X}| \leq |\mathbf{Y}|$ , the *optimal partial matching*  $\pi^*$  pairs each point in  $\mathbf{X}$  to some unique point in  $\mathbf{Y}$  such that the total distance between matched points is minimized:  $\pi^* = \arg \min_{\pi} \sum_{\mathbf{x}_i \in \mathbf{X}} \|\mathbf{x}_i - \mathbf{y}_{\pi_i}\|_1$ , where  $\pi_i$  specifies which point  $\mathbf{y}_{\pi_i}$  is matched to  $\mathbf{x}_i$ , and  $\|\cdot\|_1$  denotes the  $L_1$  norm. For sets with  $m$  features, the Hungarian algorithm computes the optimal match in  $O(m^3)$  time, which

**Figure 3. An example pyramid match.**

Here, two 1-D feature sets are used to form two histogram pyramids. Each row corresponds to a pyramid level. In (a), set  $Y$  is on the left, and set  $Z$  is on the right; points are distributed along the vertical axis. Light lines are bin boundaries, bold dashed lines indicate a new pair matched at this level, and bold solid lines indicate a match already formed at a finer resolution level. In (b) multiresolution histograms are shown; (c) shows their intersections. The pyramid match function  $\mathcal{P}_\Delta$  uses these intersection counts to measure how many new matches occurred at each level. Here,  $\mathcal{I}_i = \mathcal{I}(H_i(Y), H_i(Z)) = 2, 4, 5$  across levels, so the number of new matches counted are 2, 2, 1. The weighted sum over these counts gives the pyramid match similarity.

The figure is reprinted from Grauman and Darrell<sup>12</sup> with permission, ©2005 IEEE.



severely limits the practicality of large input sizes. In contrast, the pyramid match approximation requires only  $O(mL)$  time, where  $L = \log D$ , and  $L \ll m$ . In practice, this translates to speedups of several orders of magnitude relative to the optimal match for sets with thousands of features.

We use a multidimensional, multi-resolution histogram pyramid to partition the feature space into increasingly larger regions. At the finest resolution level in the pyramid, the partitions (bins) are very small; at successive levels they continue to grow in size until a single bin encompasses the entire feature space. At some level along this gradation in bin sizes, any two particular points from two given point sets will begin to share a bin in the pyramid, and when they do, they are considered matched. The key is that the pyramid allows us to extract a matching score without computing distances between any of the points in the input sets—the size of the bin that two points share indicates the farthest distance they could be from one another. We show that a weighted intersection of two pyramids defines an implicit partial correspondence based on the smallest histogram cell where a matched pair of points first appears.

Let a histogram pyramid for input example  $X \in S$  be defined as:  $\Psi(X) = [H_0(X), \dots, H_{L-1}(X)]$ , where  $L$  specifies the number of pyramid levels, and  $H_i(X)$  is a

histogram vector over points in  $X$ . The bins continually increase in size from the finest-level histogram  $H_0$  until the coarsest-level histogram  $H_{L-1}$ . For low-dimensional feature spaces, the boundaries of the bins are computed simply with a uniform partitioning along all feature dimensions, with the length of each bin side doubling at each level. For high-dimensional feature spaces (for example,  $d > 15$ ), we use hierarchical clustering to concentrate the bin partitions where feature points tend to cluster for typical point sets.<sup>13</sup> In either case, we maintain a sparse representation per point set that maps each point to its bin indices. Even though there is an exponential growth in the number of possible histogram bins relative to the feature dimension (for uniform bins) or histogram levels (for nonuniform bins), any given set of features can occupy only a small number of them. An image with  $m$  features results in a pyramid description with no more than  $mL$  nonzero entries to store.

Two histogram pyramids implicitly encode the correspondences between their point sets, if we consider two points matched once they fall into the same histogram bin, starting at the finest resolution level. The matching is a hierarchical process: vectors not found to correspond at a fine resolution have the opportunity to be matched at coarser resolutions.

Thus, for each pyramid level, we want to count the number of “new” matches—the number of feature pairs that were not in correspondence at any finer resolution level. For example, in Figure 3, there are two points matched at the finest scale, two new matches at the medium scale, and one at the coarsest scale.

To calculate the match count, we use histogram intersection, which measures the “overlap” between the mass in two histograms:  $\mathcal{I}(P, Q) = \sum_{j=1}^r \min(P_j, Q_j)$ , where  $P$  and  $Q$  are histograms with  $r$  bins, and  $P_j$  denotes the count of the  $j$ -th bin. The intersection value effectively counts the number of points in two sets that match at a given quantization level. To calculate the number of newly matched pairs induced at level  $i$ , we only need to compute the difference between successive levels’ intersections. By using the change in intersection values at each level, we count matches without ever explicitly searching for similar points or computing inter-feature distances.

The pyramid match similarity score  $\mathcal{P}_\Delta$  between two input sets  $Y$  and  $Z$  is then defined as the weighted sum of the number of new matches per level:

$$\mathcal{P}_\Delta(\Psi(Y), \Psi(Z)) = \sum_{i=0}^{L-1} w_i (\mathcal{I}(H_i(Y), H_i(Z)) - \mathcal{I}(H_{i-1}(Y), H_{i-1}(Z))).$$

The number of new matches induced at level  $i$  is weighted by  $w_i = \frac{1}{d^2}$  to reflect the (worst-case) similarity of points matched at that level. This weighting reflects a geometric bound on the maximal distance between any two points that share a particular bin. Intuitively, similarity between vectors at a finer resolution—where features are more distinct—is rewarded more heavily than similarity between vectors at a coarser level.

We combine the scores resulting from multiple pyramids with randomly shifted bins in order to alleviate quantization effects, and to enable formal error bounds. The approximation error for the pyramid match cost is bounded in the expectation by a factor of  $C \cdot d \log D$ , for a constant  $C$ .<sup>15</sup> We have also proven that the pyramid match kernel (PMK) naturally forms a Mercer kernel, which essentially means that it satisfies the necessary technical requirements to permit its use as a similarity function within a number of existing kernel-based machine learning methods.

Previous approximation methods have also considered a hierarchical decomposition of the feature space to reduce complexity<sup>1,2,5,17</sup>; the method by Indyk and Thaper<sup>17</sup> particularly inspired our approach. However, earlier matching approximations assume equally sized input sets, and cannot compute partial matches. In addition, while previous techniques suffer from distortion factors that are linear in the feature dimension, we have shown how to alleviate this decline in accuracy by tuning the hierarchical decomposition according to the particular structure of the data.<sup>13</sup> Finally, our approximation is unique in that it forms a valid Mercer kernel, and is useful in the context of various learning applications.

In short, the pyramid match gives us an efficient way to measure the similarity between two images based on the matching between their (potentially many) local features. Now, given a query image such as the one on the left of Figure 1, we can first extract descriptors for its local regions using any standard feature extractor,<sup>23,30</sup> and then find its relevant “neighbors” in the collection on the right by computing and sorting their pyramid match scores. In this way, we are able to

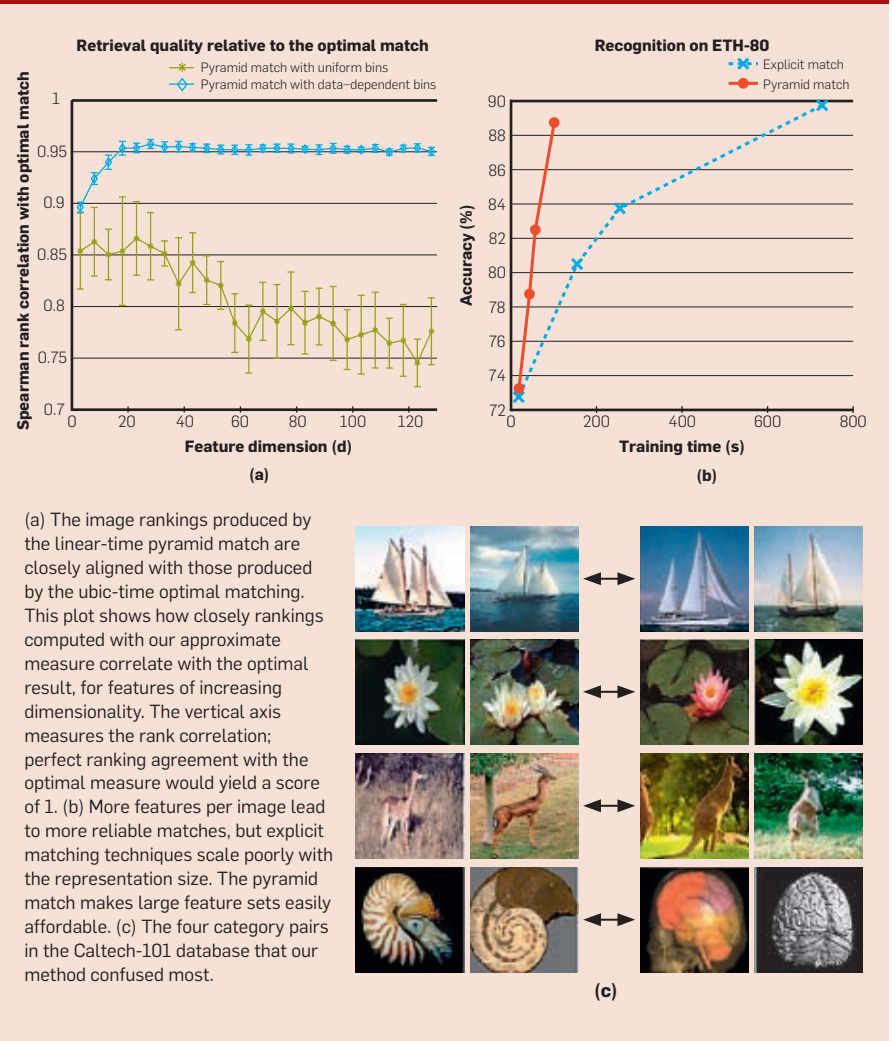
search the image collection based on content alone.

Figure 4 shows some illustrative results using two well-known publicly available benchmark datasets, the ETH-80 and Caltech-101. Both datasets are used to measure image categorization accuracy. The ETH collection is comprised of 80 object instances from eight different categories positioned on simple backgrounds; it is among the first benchmarks established for the categorization task, and since several categories are visually rather similar (for example, horse and cow, apple and tomato), it is a good test for detailed discrimination. The Caltech collection, first introduced in 2003, contains 101 categories. It is challenging due to the magnitude of the multiclass problem it poses, and for many categories it offers noticeable intraclass appearance variation. It has received much attention in the

research community and today stands as a key point of comparison for existing methods. For all the following results, we use the SIFT descriptor,<sup>23</sup> which is insensitive to shifts and rotations in the image yet provides a distinctive summary of a local patch.

The leftmost plot of Figure 4 demonstrates that when the pyramid match is used to sort the images from the ETH-80 in a retrieval task, its complete ranking of the database examples is highly correlated to that of the optimal matching. The vertical axis measures how well results from two variants of the PMK agree with the optimal cubic-time results, and the horizontal axis shows the relative impact of the feature dimension  $d$ . While for low-dimensional features either a uniform or data-dependent partitioning of the feature space is adequate for good results, due to the curse of dimensionality, a data-dependent pyramid bin

**Figure 4. The image rankings produced by the linear-time pyramid match.**





structure is much more effective for high-dimensional features.

The center plot shows accuracy as a function of computation time when the eight categories of the same dataset are learned using local feature matches between images. The plot compares the performance of the pyramid match to an exact matching function that averages the cost between the closest features in one set to the other. The horizontal axis measures the total training time, which is directly affected by the size of the feature sets. To vary the size of a typical set, we tune the saliency parameter controlling how many regions are detected per image. For both methods, more features lead to striking accuracy improvements; this behavior is expected since introducing more features assures better coverage of all potentially relevant image regions. However, the linear-time pyramid match offers a key advantage in terms of computational cost, reaching peak performance for significantly less computation time.

On the Caltech-101 benchmark, we have shown that classifiers employing the PMK with a variety of features currently yield one of the most accurate results in the field,<sup>20</sup> with 74% accuracy on the 101-way decision problem when training with just 15 exemplars per class. Figure 4c shows example images from four pairs of categories in the Caltech-101 dataset that cause the most confusion for the pyramid match: schooner and ketch, lotus and

water lily, gerenuk and kangaroo, and nautilus and brain. In each row, the two images on the left have local features that match quite well to the two on the right, as compared to images from any of the other 100 classes in the dataset. Some of these confused category pairs have rather subtle distinctions in appearance. However, the case of the gerenuk and kangaroo reveals a limitation of the completely local description, as by definition it fails to capture the significance of the global contour shapes of the two objects.

Overall, approaches based on the pyramid match consistently show accuracy that is competitive with (or better than) the state of the art while requiring dramatically less computation time. This complexity advantage frees us to consider much richer representations than were previously practical. Methods that compute explicit correspondences require about one minute to match a novel example; in the time that these methods recognize a single object, the pyramid match can recognize several hundred.<sup>15</sup> Due to its flexibility and efficiency, the pyramid match has been adapted and extended for use within a number of tasks, such as scene recognition,<sup>22</sup> video indexing,<sup>6</sup> human action recognition,<sup>24</sup> and robot localization.<sup>25</sup>

### Learning Image Metrics

Thus far, we have considered how to robustly measure image similarity

in situations where we have no background knowledge; that is, where the system only has access to the image content itself. However, in many cases the system could also receive external side-information that might benefit its comparisons. For example, if provided with partially annotated image examples, or if a user wants to enforce similarity between certain types of images, then we ought to use those constraints to adapt the similarity measure.

A good distance metric between images accurately reflects the true underlying relationships. It should report small distances for examples that are similar in the parameter space of interest (or that share a class label), and large distances for unrelated examples. Recent advances in metric learning make it possible to learn distance functions that are more effective for a given problem, provided some partially labeled data or constraints are available (see Yang<sup>32</sup> and references within). By taking advantage of the prior information, these techniques offer improved accuracy. Typically, the strategy is to optimize any parameters to the metric so as to best satisfy the desired constraints.

Figure 5a depicts how metric learning can influence image comparisons: the similarity (solid line) and dissimilarity (dotted lines) constraints essentially warp the feature space to preserve the specified relationships, and generalize to affect distances between other examples like them. In

**Figure 5. The learned metric.**



(a) By constraining some examples to be similar (green solid line), and others to be dissimilar (red dotted lines), the learned metric refines the original distance function so that examples are close only when they share the relevant features. (b) Retrieval accuracy is improved by replacing two matching-based metrics (PMK and CORR) with their learned counterparts (ML + PMK and ML + CORR).

Plot reprinted from Jain et al.<sup>19</sup> with permission, ©2008 IEEE.

this illustrative example, even though we may measure high similarity between the greenery portions of both the cockatoo and koala images, the dissimilarity constraint serves to refocus the metric on the other (distinct) parts of the images.

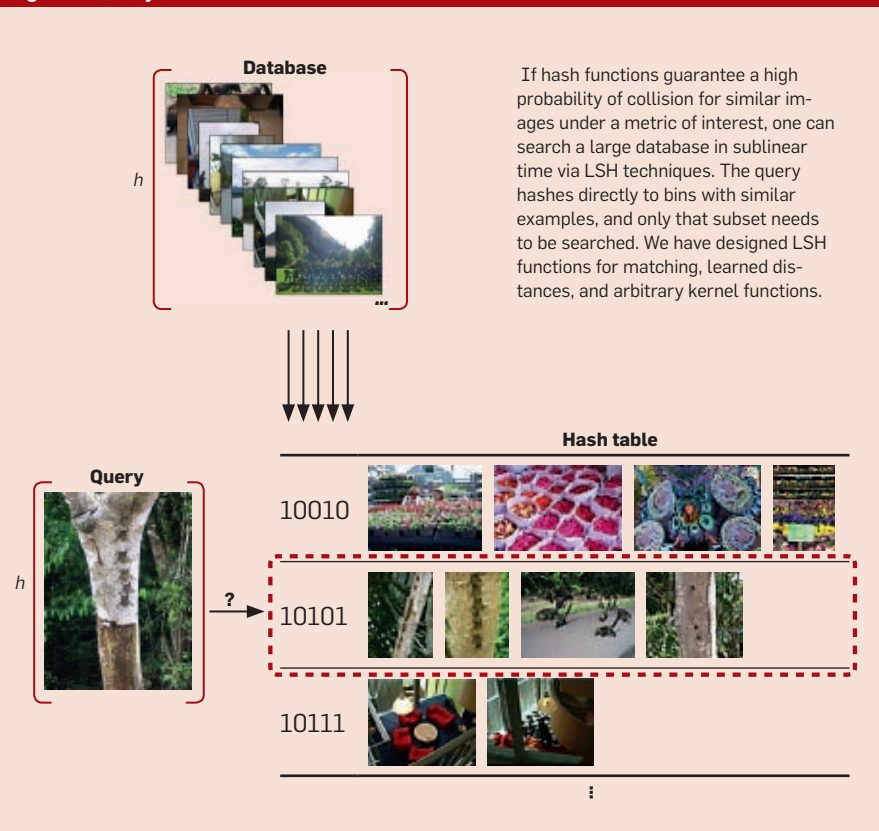
In the case of the pyramid match, the weights associated with matches at different pyramid levels can be treated as learnable parameters. While fixing the weights according to the bin diameters gives the most accurate approximation of true inter-feature distances in a geometric sense, when we have some annotated images available, we can directly learn the weights that will best map same-class images close together.<sup>18</sup>

The idea is that the best matching function is specific to the class of images, and is inherently defined by the variability a given class exhibits. For example, to distinguish one skyscraper from another, we might expect same-class examples to contain some very tight local feature correspondences, whereas to distinguish all skyscrapers from koalas, we expect feature matches to occur at greater distances even among same-class examples. While the same *type* of image feature may be equally relevant in both situations, what is unique is the *distance* at which similarity is significant for that feature. Therefore, by learning the reward (weight) associated with each matching level in the pyramid, we can automatically determine how close feature matches must be in order to be considered significant for a given object class.

To achieve this intuition, we observe that the PMK can be written as a weighted sum of base kernels, where each base kernel is the similarity computed at a given bin resolution. We thus can compute the weights using a form of kernel alignment,<sup>3</sup> where we find the optimal combination of kernel matrices that most closely mimics the “ideal” kernel on the training data, that is, the one that gives maximal similarity values for in-class examples and minimal values for out-of-class examples (see Jain et al.<sup>18</sup> for details).

We have also shown how image retrieval can benefit from learning the *Mahalanobis parameterization* for several distinct base metrics, including

Figure 6. Query hashes.



If hash functions guarantee a high probability of collision for similar images under a metric of interest, one can search a large database in sublinear time via LSH techniques. The query hashes directly to bins with similar examples, and only that subset needs to be searched. We have designed LSH functions for matching, learned distances, and arbitrary kernel functions.

matching functions.<sup>19</sup> Given points  $\{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ , with  $\mathbf{x}_i \in \mathcal{R}^d$ , a positive-definite  $d \times d$  matrix  $\mathbf{A}$  parameterizes the squared Mahalanobis distance:

$$d_A(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i - \mathbf{x}_j)^T \mathbf{A} (\mathbf{x}_i - \mathbf{x}_j). \quad (1)$$

A generalized inner product measures the pairwise similarity associated with that distance:  $S_A(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{A} \mathbf{x}_j$ . Thus for a kernel  $K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$ , the parameters transform the inner product in the implicit feature space as  $\phi(\mathbf{x}_i)^T \mathbf{A} \phi(\mathbf{x}_j)$ . Given a set of inter-example distance constraints, one can directly *learn* a matrix  $\mathbf{A}$  to yield a measure that is more accurate for a given problem. We use the efficient method of Davis et al.<sup>7</sup> because it is kernelizable. This method optimizes the parameters of  $\mathbf{A}$  so as to minimize how much that matrix diverges from an initial user-provided “base” parameterization, while satisfying constraints that require small distances between examples specified as similar, and large distances between pairs specified as dissimilar.

Figure 5b shows the significant retrieval accuracy gains achieved when we learn image metrics using

two matching-based kernels as the base metrics. The first kernel is the PMK, the approximate matching measure defined here. The second kernel is defined in Zhang et al.,<sup>33</sup> and uses exhaustive comparisons between features to compute a one-to-many match based on both descriptor and positional agreement; we refer to it as CORR for “correspondence.” For this dataset of 101 object types, note that chance performance would yield an accuracy rate of only 1%. Both base metrics do the most they can by matching the local image features; the learned parameters adapt those metrics to better reflect the side-information specifying a handful of images from each class that ought to be near (or far) from the others.

### Searching Image Collections in Sublinear Time

Now that we have designed effective similarity measures, how will image search scale? We must be able to use these metrics to query a very large image database—potentially on the order of millions of examples or more. Certainly, a naive linear scan that compares the query against every database image is

not feasible, even if the metric itself is efficient. Unfortunately, traditional methods for fast search cannot guarantee low query-time performance for arbitrary specialized metrics.<sup>3</sup> This section overviews our work designing hash functions that enable approximate similarity search for both types of metrics introduced above: a matching between sets, and learned Mahalanobis kernels.

The main idea of our approach is to construct a new family of hash functions that will satisfy the *locality sensitivity* requirement that is central to existing randomized algorithms<sup>5, 16</sup> for approximate nearest neighbor search. Locality sensitive hashing (LSH) has been formulated in two related contexts—one in which the likelihood of collision is guaranteed relative to a threshold on the radius surrounding a query point,<sup>16</sup> and another where collision probabilities are equated with a similarity function score.<sup>5</sup> We use the latter definition here.

A family of LSH functions  $\mathcal{F}$  is a distribution of functions where for any two objects  $\mathbf{x}_i$  and  $\mathbf{x}_j$ ,

$$\Pr_{h \in \mathcal{F}}[h(\mathbf{x}_i) = h(\mathbf{x}_j)] = \text{sim}(\mathbf{x}_i, \mathbf{x}_j), \quad (2)$$

where  $\text{sim}(\mathbf{x}_i, \mathbf{x}_j) \in [0, 1]$  is some similarity function, and  $h(\mathbf{x})$  is a hash function drawn from  $\mathcal{F}$  that returns a single bit.<sup>5</sup> Concatenating a series of  $b$  hash functions drawn from  $\mathcal{F}$  yields  $b$ -dimensional hash keys. When  $h(\mathbf{x}_i) = h(\mathbf{x}_j)$ ,  $\mathbf{x}_i$  and  $\mathbf{x}_j$  collide in the hash table. Because the probability that two inputs collide is equal to the similarity between them, highly similar objects are indexed together in the hash table with high probability. On the other hand, if two objects are very dissimilar, they are unlikely to share a hash key (see Figure 6). Given valid LSH

<sup>a</sup> Data structures based on spatial partitioning and recursive decomposition have been developed for faster search, e.g.,  $k$ - $d$  trees<sup>9</sup> and metric trees.<sup>31</sup> While their expected query time requirement may be logarithmic in the database size, selecting useful partitions can be expensive and requires good heuristics; worse, in high-dimensional spaces all exact search methods are known to provide little query time improvement over a naive linear scan.<sup>16</sup> The expected query time for a  $k$ - $d$  tree contains terms that are exponential in the dimension of the features,<sup>9</sup> making them especially unsuitable for the pyramid representation where the dimension can be on the order of millions.

functions, the query time for retrieving  $(1 + \varepsilon)$ -near neighbors is bounded by  $O(N^{1/(1+\varepsilon)})$  for the Hamming distance and a database of size  $N$ .<sup>10</sup> One can therefore trade off the accuracy of the search with the query time required.

Note that Equation 2 is essentially a gateway to LSH: if one can provide a distribution of hash functions guaranteed to preserve this equality *for the similarity function of interest*, then approximate nearest neighbor search may be performed in sublinear time. Existing LSH functions can accommodate the Hamming distance,  $L_p$  norms, and inner products, and such functions have been explored previously in the vision community. In the following we show how to enable sublinear time search with LSH for metrics that are particularly useful for image search.

**Matching-sensitive hashing.** Even though the pyramid match makes each individual matching scalable relative to the number of features per image, once we want to search a large database of images according to the correspondence-based distance, we still cannot afford a naive linear scan. To guarantee locality sensitivity for a matching, we form an embedding function that maps our histogram pyramids into a vector space in such a way that the inner product between vectors in that space exactly yields the PMK similarity value.<sup>14</sup>

This remapping is motivated by the fact that randomized hash functions exist for similarity search with the inner product.<sup>5</sup> Specifically, Goemans and Williamson<sup>11</sup> show that the probability that a hyperplane  $\mathbf{r}$  drawn uniformly at random separates two vectors  $\mathbf{x}_i$  and  $\mathbf{x}_j$  is directly proportional to the angle between them:  $\Pr[\text{sgn}(\mathbf{r}^T \mathbf{x}_i) \neq \text{sgn}(\mathbf{r}^T \mathbf{x}_j)] = \frac{1}{\pi} \cos^{-1}(\mathbf{x}_i^T \mathbf{x}_j)$ . An LSH function that exploits this relationship is given by Charikar.<sup>5</sup> The hash function  $h_r$  accepts a vector  $\mathbf{x} \in \mathbb{R}^d$ , and outputs a bit depending on the sign of its product with  $\mathbf{r}$ :

$$h_r(\mathbf{x}) = \begin{cases} 1, & \text{if } \mathbf{r}^T \mathbf{x} \geq 0 \\ 0, & \text{otherwise} \end{cases} \quad (3)$$

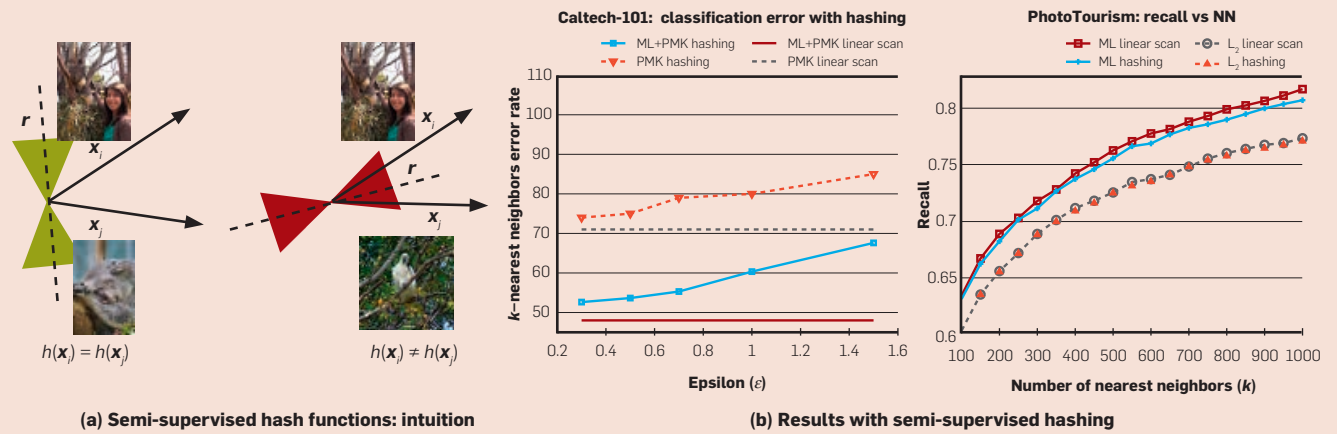
Since  $\Pr[h_r(\mathbf{x}_i) = h_r(\mathbf{x}_j)] = 1 - \frac{1}{\pi} \cos^{-1}(\mathbf{x}_i^T \mathbf{x}_j)$ , the probability of collision is high whenever the examples' inner product is high.<sup>5</sup>

To embed the pyramid match as an inner product, we exploit the relationship between a dot product and the min operator used in the PMK's intersections. Taking the minimum of two values is equivalent to computing the dot product of a unary-style encoding in which a value  $v$  is written as a list of  $v$  ones, followed by a zero padding large enough to allot space for the maximal value that will occur. So, since a weighted intersection value is equal to the intersection of weighted values, we can compute the embedding by stacking up the histograms from a single pyramid, and weighting the entries associated with each pyramid level appropriately. Our embedding enables LSH for normalized partial match similarity with local features, and we have shown that it can achieve results very close to a naive linear scan when searching only a small fraction of an image database (1%–2%) (see Grauman and Darrell<sup>14</sup> for more details).

**Semi-supervised hashing.** To provide suitable hash functions for *learned* Mahalanobis metrics, we propose altering the distribution from which the randomized hyperplanes are drawn. Rather than drawing the vector  $\mathbf{r}$  uniformly at random, we want to bias the selection so that similarity constraints provided for the metric learning process are also respected by the hash functions. In other words, we still want similar examples to collide, but now that similarity cannot be purely based on the image measurements  $\mathbf{x}_i$  and  $\mathbf{x}_j$ ; it must also reflect the constraints that yield the improved (learned) metric (see Figure 7a). We refer to this as “semi-supervised” hashing, since the hash functions will be influenced by any available partial annotations, much as the learned metrics were in the previous section.

In Jain et al.,<sup>19</sup> we present a straightforward solution for the case of relatively low-dimensional input vector spaces, and further derive a solution to accommodate very high-dimensional data for which explicit input space computations are infeasible. The former contribution makes fast indexing accessible for numerous existing metric learning methods, while the latter is of particular interest for commonly used image representations, such as bags-of-words or multiresolution histograms.



**Figure 7. Semi-supervised hash functions.**


(a) A generic hash function would choose the orientation of the hyperplane  $r$  uniformly at random, causing collisions only for examples that have small angles between their features ( $x_i$  and  $x_j$ ). In contrast, the distribution of our randomized semi-supervised hash functions is such that examples like those constrained to be similar are more likely to collide (left), while pairs like those constrained to be dissimilar are less likely to collide (right). Here the hourglass shapes denote the regions from which our randomized hash functions will most likely be drawn. (b) Semi-supervised hash functions encode the learned metric, and allow guaranteed sublinear time queries that are similar in accuracy to a naive linear scan. Plots are reprinted from Jain et al.<sup>19</sup> with permission, ©2008 IEEE.

Given the matrix  $A$  for a metric learned as above, such that  $A = G^T G$ , we generate the following randomized hash functions  $h_{r,A}$ :

$$h_{r,A}(x) = \begin{cases} 1, & \text{if } r^T G x \geq 0 \\ 0, & \text{otherwise} \end{cases} \quad (4)$$

where the vector  $r$  is chosen at random from a  $d$ -dimensional Gaussian distribution with zero mean and unit variance.

By parameterizing the hash functions by both  $r$  and  $G$ , we enforce the following probability of collision:

$$\Pr[h_{r,A}(x_i) = h_{r,A}(x_j)] = 1 - \frac{1}{\pi} \cos^{-1} \left( \frac{x_i^T A x_j}{\sqrt{|G x_i| |G x_j|}} \right),$$

which sustains the LSH requirement for a learned Mahalanobis metric. Essentially we have shifted the random hyperplane  $r$  according to  $A$ , and by factoring it by  $G$  we allow the random hash function itself to “carry” the information about the learned metric. The denominator in the cosine term normalizes the kernel values.

For low-dimensional data, we could equivalently transform all the data according to  $A$  prior to hashing. However, the matrix  $A$  has  $d^2$  entries, and thus for very high-dimensional input spaces it cannot be represented

explicitly, and we must work in the implicit kernel space. For example, for features like the histogram pyramids above, we have  $d = 10^6$ . The examples are sparse and representable; however, the matrix  $A$  is dense and is not. This complicates the computation of hash functions, as they can no longer be computed directly as in Equation 4. To handle this, we derived an algorithm that simultaneously makes implicit updates to both the hash functions and the metric being learned. We show it is possible to compute the value of  $r^T G$  indirectly, based on comparisons between the points involved in similarity constraints and the new example  $x$  that we want to hash. See Jain et al.<sup>19</sup> for details.

Figure 7b shows results using our semi-supervised hash functions. In the left-hand plot, we see the learned metric (denoted ‘ML’) significantly improves the base metric in the image retrieval task for the Caltech data. Additionally, we now can offer sublinear time search even once the metric has been altered by the input similarity constraints. Note how accuracy varies as a function of  $\epsilon$ , the parameter controlling how many examples we have to search per query; the more examples we can afford to search, the stronger our guarantee of approximating an exhaustive linear scan.

The right-hand plot shows results using another database, 300K patches

from the PhotoTourism project.<sup>28</sup> Here  $L_2$  is the base metric; the recall rate is substantially improved once we learn a metric on top of it. Negligible accuracy is sacrificed when searching with our semi-supervised hash functions (as seen by the closeness of the top two curves), yet our hashing strategy requires touching only 0.8% of the patches in the database. In our MATLAB implementation, we observe speedup factors of about 400 relative to a linear scan for databases containing half a million examples. Due to the query-time guarantees our hash functions enable, that factor grows rapidly with the size of the database.

Most recently, we have derived hash functions to enable fast search with arbitrary kernel functions.<sup>21</sup>

Relative to traditional exact search data structures, the approximate hashing approach is critical to performance when inputs are high-dimensional. Modifications to classic tree structures have also been explored to improve search time with high-dimensional image features;<sup>4,26</sup> however, such approaches do not provide query-time guarantees, and are not applicable to searching with learned metrics. By hashing to buckets containing a collection of examples with a high probability of being very similar to the query, we are able to sort out the most relevant list of near neighbors. This is important for content-based retrieval, where we

do not expect the single nearest exemplar to answer the query, but rather that the pool of nearby content will give the user and/or downstream processes access to relevant candidates.

## Conclusion

As the world's store of digital images continues to grow exponentially, and as novel data-rich approaches to computer vision begin to emerge, fast techniques capable of accurately searching very large image collections are critical. The algorithms we have developed aim to provide robust but scalable image search, and results show the practical impact. While motivated by vision problems, these methods are fairly general, and may be applicable in other domains where rich features and massive data collections abound, such as computational biology or text processing.


Looking forward, an important challenge in this research area is to develop the representations that will scale in terms of their distinctiveness; once the space of images is even more densely populated, relative differences are subtle. At the same time, flexibility is still a key to handling intra-category variation. While our search methods can guarantee query-time performance, it is not yet possible to guarantee a level of discrimination power for the features chosen. In addition, a practical issue for evaluating algorithms in this space is the difficulty of quantifying accuracy for truly massive databases; the data itself is easy to come by, but without ground truth annotations, it is unclear how to rigorously evaluate performance.

An interesting aspect of the image search problem is the subjectivity related to a real user's perception of the quality of a retrieval. We can objectively quantify accuracy in terms of the categories contained in a retrieved image, which is helpful to systematically validate progress. Moreover, example-based search often serves as one useful stage in a larger pipeline with further processing downstream. Nonetheless, when end users are in the loop, the perception of quality may vary. On the evaluation side, this uncertainty could be addressed by collecting user appraisals of similarity, as is more standard in natural language processing. In terms of the algorithms themselves, however, one can also exploit classic feedback

and query-refinement devices to tailor retrieval toward the current user. For example, we could construct learned image metrics with constraints that target the preferences of a given user or group of users.

We are currently exploring online extensions to our algorithms that allow similarity constraints to be processed in an incremental fashion, while still allowing intermittent queries. We are also pursuing active learning methods that allow the system to identify which image annotations seem most promising to request, and thereby most effectively use minimal manual input.

## Acknowledgments

I am fortunate to have worked with a number of terrific collaborators throughout the various stages of the projects overviewed in this article—in particular, Trevor Darrell, Prateek Jain, and Brian Kulis. This research was supported in part by NSF CAREER IIS-0747356, Microsoft Research, and the Henry Luce Foundation. I would like to thank Kathryn McKinley and Yong Jae Lee for feedback on previous drafts, as well as the anonymous reviewers for their helpful comments. Thanks to the following Flickr users for sharing their photos under the Creative Commons license: belgian-chocolate, c.j.b., edwinn.11, piston9, staminaplus100, rick-yrhodes, Rick Smit, Krikrit, Vanessa Pike-Russell, Will Ellis, Yvonne in Willowick Ohio, robertpaulyoung, lin padgham, tkcrash123, jennifrog, Zemzina, Irene2005, and CmdrGravy. 

## References

1. Agarwal, P., Varadarajan, K.R. A near-linear algorithm for Euclidean bipartite matching. In *Symposium on Computational Geometry* (2004).
2. Avis, D. A survey of heuristics for the weighted matching problem. *Networks*, 13 (1983), 475–493.
3. Bach, F., Lanckriet, G., Jordan, M. Multiple kernel learning, conic duality, and the SMO algorithm. In *International Conference on Machine Learning* (2004).
4. Beis, J., Lowe, D. Shape indexing using approximate nearest-neighbour search in high dimensional spaces. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (1997).
5. Charikar, M. Similarity estimation techniques from rounding algorithms. In *ACM Symposium on Theory of Computing* (2002).
6. Choi, J., Jeon, W., Lee, S.-C. Spatio-temporal pyramid matching for sports videos. In *Proceedings of the ELACM Conference on Multimedia Information Retrieval* (2008).
7. Davis, J., Kulis, B., Jain, P., Sra, S., Dhillon, I. Information-theoretic metric learning. In *International Conference on Machine Learning* (2007).
8. Flickner, M. et al. Query by image and video content:

- The QBIC system. *IEEE Comput.* 28, 9 (1995), 23–32.
9. Friedman, J., Bentley, J., Finkel, R. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw.* 3, 3 (1977), 209–226.
10. Gionis, A., Indyk, P., Motwani, R. Similarity search in high dimensions via hashing. In *Proceedings of the International Conference on Very Large Data Bases* (1999).
11. Goemans, M., Williamson, D. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. Assoc. Comput. Mach.* 42, 6 (1995), 1115–1145.
12. Grauman, K., Darrell, T. The pyramid match kernel: discriminative classification with sets of image features. In *Proceedings of the IEEE International Conference on Computer Vision* (2005).
13. Grauman, K., Darrell, T. Approximate correspondences in high dimensions. In *Advances in Neural Information Processing Systems* (2006).
14. Grauman, K., Darrell, T. Pyramid match hashing: Sub-linear time indexing over partial correspondences. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2007).
15. Grauman, K., Darrell, T. The pyramid match kernel: efficient learning with sets of features. *J. Mach. Learn. Res.* 8 (2007), 725–760.
16. Indyk, P., Motwani, R. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Symposium on Theory of Computing* (1998).
17. Indyk, P., Thaper, N. Fast image retrieval via embeddings. In *International Workshop on Statistical and Computational Theories of Vision* (2003).
18. Jain, P., Huynh, T., Grauman, K. *Learning Discriminative Matching Functions for Local Image Features*. Technical Report, UT-Austin, April 2007.
19. Jain, P., Kulis, B., Grauman, K. Fast image search for learned metrics. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2008).
20. Kapoor, A., Grauman, K., Urtasun, R., Darrell, T. Gaussian processes for object categorization. *Int. J. Comput. Vis.* 88, 2 (2009), 169–188.
21. Kulis, B., Grauman, K. Kernelized locality-sensitive hashing for scalable image search. In *Proceedings of the IEEE International Conference on Computer Vision* (2009).
22. Lazebnik, S., Schmid, C., Ponce, J. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2006).
23. Lowe, D. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vis.* 60, 2 (2004).
24. Lv, F., Nevatia, R. Single view human action recognition using key pose matching and Viterbi path searching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2007).
25. Murillo, A. et al. From omnidirectional images to hierarchical localization. *Robotics Auton. Syst.* 55, 5 (May 2007), 372–382.
26. Nister, D., Stewenius, H. Scalable recognition with a vocabulary tree. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2006).
27. Pinz, A. Object categorization. *Found. Trend. Comput. Graph. Vis.* 1, 4 (2006), 255–353.
28. Snavely, N., Seitz, S., Szeliski, R. PhotoTourism: Exploring photos in 3D. In *SIGGRAPH* (2006).
29. Swain, M., Ballard, D. Color indexing. *Int. J. Comput. Vis.* 7, 1 (1991), 11–32.
30. Tuytelaars, T., Mikolajczyk, K. Local invariant feature detectors: A survey. *Found. Trend. Comput. Graph. Vis.* 3, 3 (2008), 177–280.
31. Uhlmann, J. Satisfying general proximity/similarity queries with metric trees. *Inf. Process. Lett.* 40 (1991), 175–179.
32. Yang, L. *Distance Metric Learning: A Comprehensive Survey*. Technical Report, Michigan State University, 2006.
33. Zhang, H., Berg, A., Maire, M., Malik, J. SVM-KNN: Discriminative nearest neighbor classification for visual category recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2006).

Kristen Grauman (grauman@cs.utexas.edu) is an assistant professor in the Department of Computer Science at the University of Texas at Austin.

# research highlights

---

P. 96

**Technical  
Perspective  
Building Confidence  
in Multicore Software**

By Vivek Sarkar

P. 97

**Asserting and Checking  
Determinism for  
Multithreaded Programs**

By Jacob Burnim and Koushik Sen

---

P. 106

**Technical  
Perspective  
Learning To Do  
Program Verification**

By K. Rustan M. Leino

P. 107

**seL4: Formal Verification  
of an Operating-System Kernel**

By Gerwin Klein, June Andronick, Kevin Elphinstone,  
Gernot Heiser, David Cock, Philip Derrin, Dhammika Elkaduwe,  
Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell,  
Harvey Tuch, and Simon Winwood



# Technical Perspective

## Building Confidence in Multicore Software

By Vivek Sarkar

SURPRISES MAY BE fun in real life, but not so in software. One approach to avoiding surprises in software is to establish its functional correctness, either by construction or by verification, but this is feasible in only a limited set of domains. Instead, the predominant method in single-threaded software development has been an iterative approach of design, coding, testing, and bug fixes. The cornerstone of this practice is *determinism*; that is, the expectation a program will exhibit the same behavior each time it is executed with the same inputs. Even if a program has a bug, it is comforting to know the buggy behavior can be reproduced by others, and workarounds can be shared until the developer provides a fix.

In this context, multithreaded programs prove to be more challenging to handle than single-threaded programs. For decades multithreaded programs were executed on single-core processors and users became accustomed to deterministic sets of behaviors exhibited by standard thread schedulers. However, the move to multicore hardware has completely changed this landscape, which is why I recommend you read the following paper.

Jacob Burnim and Koushik Sen propose an *assertion framework* for specifying regions of multithreaded software that are expected to behave deterministically, and describe a runtime library for checking these assertions that is guaranteed to be sound. Though the proposed runtime checking is incomplete in general, preliminary evaluations suggest that this approach can be effective in identifying many common sources of nondeterminism. Runtime assertion-checking is used in many situations nowadays, including null-pointer, array-bounds checks, and type checks in managed runtimes. So, it is reasonable to view the checking assertions related to de-


terminism as a natural future extension to the runtime checks performed in modern software.

A key challenge in multicore programs is that determinism lies in the eye of the beholder. If two executions of the same program with the same input produce outputs that are non-identical but semantically equivalent, should the program be considered deterministic or not? For example, consider a program that produces two different floating-point values in two executions with the same input. From the viewpoint of a strict definition of determinism, the program is unquestionably nondeterministic. From the viewpoint of a more relaxed definition in which all values within a certain error threshold are permissible as outputs, the program may well be considered to be deterministic. A similar situation arises for programs that produce (say) ordered linked lists as data structure representations of unordered sets. Two non-identical outputs may still be considered equivalent if they contain the same set of elements, albeit in different orders.

Given this range of interpretations for determinism, it isn't obvious how assertions for determinism should be formulated. The approach taken in this paper is to extend the concepts of preconditions ("assume" clauses) and

**A key challenge in multicore programs is that determinism lies in the eye of the beholder.**

postconditions ("assert" clauses) by using *bridge predicates*. A bridge predicate relates values arising from two different executions of the same program, thereby providing the foundation for asserting *semantic determinism* at any desired level of user-specified granularity. One of the examples discussed is parallel matrix multiply, where the bridge predicate in the precondition assumes the input matrices from two executions differ entry-by-entry by no more than an error threshold, and the bridge predicate in the postcondition asserts that a similar property holds for the output matrices. Note that these assertions are focused on determinism and not on functional correctness. For example, a functionally incorrect implementation of parallel matrix multiply that returns the identity matrix for all inputs will always pass determinism checking.

At this point I hope I've raised a number of questions in your mind. Can the determinism assertions be generated automatically? What is the relationship between checking assertions for determinism and detection of data races? Are there any assumptions made about the underlying system software and hardware, such as the memory consistency model? Can the programming constructs advocated by transactional memory researchers help address this problem? Are there applications of determinism assertions to single-threaded programs? If you're interested in any of these questions, you need to read the following paper to better understand the ramifications of parallel hardware on determinism guarantees in multithreaded software! 

**Vivek Sarkar** (vsarkar@rice.edu) is an ACM Fellow and a professor of computer science and of electrical and computer engineering at Rice University, where he holds the E.D. Butcher Chair in Engineering.

© 2010 ACM 0001-0782/10/0600 \$10.00

# Asserting and Checking Determinism for Multithreaded Programs

By Jacob Burnim and Koushik Sen

## Abstract

**The trend towards processors with more and more parallel cores is increasing the need for software that can take advantage of parallelism. The most widespread method for writing parallel software is to use explicit threads. Writing correct multithreaded programs, however, has proven to be quite challenging in practice. The key difficulty is *nondeterminism*. The threads of a parallel application may be interleaved nondeterministically during execution. In a buggy program, nondeterministic scheduling can lead to nondeterministic results—where some interleavings produce the correct result while others do not.**

**We propose an assertion framework for specifying that regions of a parallel program behave deterministically despite nondeterministic thread interleaving. Our framework allows programmers to write assertions involving pairs of program states arising from different parallel schedules. We describe an implementation of our deterministic assertions as a library for Java, and evaluate the utility of our specifications on a number of parallel Java benchmarks. We found specifying deterministic behavior to be quite simple using our assertions. Further, in experiments with our assertions, we were able to identify two races as true parallelism errors that lead to incorrect nondeterministic behavior. These races were distinguished from a number of benign races in the benchmarks.**

## 1. INTRODUCTION

The semiconductor industry has hit the power wall—performance of general-purpose single-core microprocessors can no longer be increased due to power constraints. Therefore, to continue to increase performance, the microprocessor industry is instead increasing the number of processing cores per die. The new “Moore’s Law” is that the number of cores will double every generation, with individual cores going no faster.<sup>2</sup>

This new trend of increasingly parallel chips means that we will have to write parallel software in order to take advantage of future hardware advances. Unfortunately, parallel software is more difficult to write and debug than its sequential counterpart. A key reason for this difficulty is *nondeterminism*—i.e., that in two runs of a parallel program on the exact same input, the parallel threads of execution can interleave differently, producing different output. Such nondeterministic thread interleaving is an essential part of harnessing the power of parallel chips, but it is a major departure from sequential programming, where we typically expect programs to behave identically in every execution on the same input. We share a

widespread belief that helping programmers manage nondeterminism in parallel software is critical in making parallel programming widely accessible.

For more than 20 years, many researchers have attacked the problem of nondeterminism by attempting to detect or predict *sources* of nondeterminism in parallel programs. The most notorious of such sources is the *data race*. A data race occurs when two threads in a program concurrently access the same memory location and at least one of those accesses is a write. That is, the two threads “race” to perform their conflicting memory accesses, so the order in which the two accesses occur can change from run to run, potentially yielding nondeterministic program output. Many algorithms and tools have been developed to detect and eliminate data races in parallel programs. (See Burnim and Sen<sup>5</sup> for further discussion and references.) Although the work on data race detection has significantly helped in finding determinism bugs in parallel programs, it has been observed that the absence of data races is not sufficient to ensure determinism.<sup>1, 8, 9</sup> Thus researchers have also developed techniques to find high-level races,<sup>1, 16, 21</sup> likely atomicity violations,<sup>9, 8, 14</sup> and other potential sources of nondeterminism. Further, such sources of nondeterminism are not always bugs—they may not lead to nondeterministic program behavior or nondeterminism may be intended. In fact, race conditions may be useful in gaining performance while still ensuring high-level deterministic behavior.<sup>3</sup>

More recently, a number of ongoing research efforts aim to make parallel programs deterministic *by construction*. These efforts include the design of new parallel programming paradigms<sup>10, 12, 13, 19</sup> and the design of new type systems, annotations, and checking or enforcement mechanisms that could retrofit existing parallel languages.<sup>4, 15</sup> But such efforts face two key challenges. First, new languages see slow adoption and often remain specific to limited domains. Second, new paradigms often include restrictions that can hinder general-purpose programming. For example, a new type system may require complex type annotations and may forbid reasonable programs whose determinism cannot be expressed in the type system.

We argue that programmers should be provided with a framework that will allow them *to express deterministic*

The original version of this paper was published in *Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, August 2009.

behaviors of parallel programs *directly* and *easily*. Specifically, we should provide an assertion framework where programmers can *directly* and *precisely* express intended deterministic behavior. Further, the framework should be flexible enough so that deterministic behaviors can be expressed more *easily* than with a traditional assertion framework. For example, when expressing the deterministic behavior of a parallel edge detection algorithm for images, we should not have to rephrase the problem as race detection; nor should we have to write a state assertion that relates the output to the input, which would be complex and time-consuming. Rather, we should simply be able to say that, if the program is executed on the same input image, then the output image remains the same regardless of how the program's parallel threads are scheduled.

In this paper, we propose such a framework for asserting that blocks of parallel code behave deterministically. Formally, our framework allows a programmer to give a specification for a block  $P$  of parallel code as:

```
deterministic assume (Pre ( $s_0, s'_0$ )) {
  P
} assert (Post ( $s, s'$ ));
```

This specification asserts the following: Suppose  $P$  is executed twice with potentially different schedules, once from initial state  $s_0$  and once from  $s'_0$  and yielding final states  $s$  and  $s'$ . Then, if the user-specified *precondition*  $\text{Pre}$  holds over  $s_0$  and  $s'_0$ , then  $s$  and  $s'$  must satisfy the user-specified *postcondition*  $\text{Post}$ .

For example, we could specify the deterministic behavior of a parallel matrix multiply with:

```
deterministic assume (|A - A'| < 10-9 and
                    |B - B'| < 10-9) {
  C = parallel_matrix_multiply_float(A, B);
} assert (|C - C'| < 10-6);
```

Note the use of primed variables  $A'$ ,  $B'$ , and  $C'$  in the above example. These variables represent the state of the matrices  $A$ ,  $B$ , and  $C$  from a different execution. Thus, the predicates that we write inside `assume` and `assert` are different from state predicates written in a traditional assertion framework—our predicates relate a pair of states from different executions. We call such predicates *bridge predicates* and assertions using bridge predicates *bridge assertions*. A key contribution of this paper is the introduction of these bridge predicates and bridge assertions.

Our deterministic assertions provide a way to specify the correctness of the parallelism in a program independently of the program's traditional functional correctness. By checking whether different program schedules can nondeterministically lead to semantically different answers, we can find bugs in a program's use of parallelism even when unable to directly specify or check functional correctness—i.e., that the program's output is correct given its input. Inversely, by checking that a parallel program behaves deterministically, we can gain confidence

in the correctness of its use of parallelism independently of whatever method we use to gain confidence in the program's functional correctness.

We have implemented our deterministic assertions as a library for the Java programming language. We evaluated the utility of these assertions by manually adding deterministic specifications to a number of parallel Java benchmarks. We used an existing tool to find executions exhibiting data and higher-level races in these benchmarks and used our deterministic assertions to distinguish between harmful and benign races. We found it to be fairly easy to specify the correct deterministic behavior of the benchmark programs using our assertions, despite being unable in most cases to write traditional invariants or functional correctness assertions. Further, our deterministic assertions successfully distinguished the two races known to lead to undesired non-determinism from the benign races in the benchmarks.

## 2. DETERMINISTIC SPECIFICATION

In this section, we motivate and define our proposal for assertions for specifying determinism.

Strictly speaking, a block of parallel code is said to be deterministic if, given any particular initial state, all executions of the code from the initial state produce the exact same final state. In our specification framework, the programmer can specify that they expect a block of parallel code, say  $P$ , to be deterministic with the following construct:

```
deterministic {
  P
}
```

This assertion specifies that if  $s$  and  $s'$  are both program states resulting from executing  $P$  under different thread schedules from some initial state  $s_0$ , then  $s$  and  $s'$  must be equal. For example, the specification:

```
deterministic {
  C = parallel_matrix_multiply_int(A, B);
}
```

asserts that for the parallel implementation of matrix multiplication in function `parallel_matrix_multiply_int`, any two executions from the same program state must reach the same program state—i.e., with identical entries in matrix  $C$ —no matter how the parallel threads are scheduled.

A key implication of knowing that a block of parallel code is deterministic is that we may be able to treat the block as sequential in other contexts. That is, although the block may have internal parallelism, a programmer (or perhaps a tool) can hopefully ignore this parallelism when considering the larger program using the code block. For example, perhaps a deterministic block of parallel code in a function can be treated as if it were a sequential implementation when reasoning about the correctness of code calling the function.



**Semantic Determinism:** The above deterministic specification is often too conservative. For example, consider a similar example, but where  $A, B$ , and  $C$  are floating-point matrices:

```
deterministic {
  C = parallel_matrix_multiply_float(A, B);
}
```

Limited-precision floating-point addition and multiplication are not associative due to rounding error. Thus, depending on the implementation, it may be unavoidable that the entries of matrix  $C$  will differ slightly depending on the thread schedule.

In order to tolerate such differences, we must relax the deterministic specification:

```
deterministic {
  C = parallel_matrix_multiply_float(A, B);
} assert(|C - C'| < 10-6);
```

This assertion specifies that, for any two matrices  $C$  and  $C'$  resulting from the execution of the matrix multiply from the same initial state, the entries of  $C$  and  $C'$  must differ by only a small quantity (i.e.,  $10^{-6}$ ).

Note that the above specification contains a predicate over two states—each from a different parallel execution of the deterministic block. We call such a predicate a *bridge predicate*, and an assertion using a bridge predicate a *bridge assertion*. Bridge assertions are different from traditional assertions in that they allow one to write a property over two program states coming from different executions whereas traditional assertions only allow us to write a property over a single program state.

Note also that such predicates need not be equivalence relations on pairs of states. In particular, the approximate equality used above is not an equivalence relation.

This relaxed notion of determinism is useful in many contexts. Consider the following example which adds in parallel two items to a synchronized set:

```
Set set = new SynchronizedTreeSet();
deterministic {
  set.add(3); || set.add(5);
} assert(set.equals(set'));
```

If `set` is represented internally as a red-black tree, then a strict deterministic assertion would be too conservative. The structure of the resulting tree, and its layout in memory, will likely differ depending on which element is inserted first, and thus different parallel executions can yield different program states.

But we can use a bridge predicate to assert that, no matter what schedule is taken, the resulting set is *semantically*

the same. That is, for objects `set` and `set'` computed by two different schedules, the `equals` method must return true because the sets must logically contain the same elements. We call this *semantic determinism*.

**Preconditions for Determinism:** So far we have described the following construct:

```
deterministic {
  P
} assert(Post);
```

where `Post` is a predicate over two program states from different executions with different thread schedules. That is, if  $s$  and  $s'$  are two states resulting from any two executions of  $P$  from the same initial state, then `Post(s, s')` holds.

The above construct could be rewritten:

```
deterministic assume(s0 = s'0) {
  P
} assert(Post);
```

That is, if any two executions of  $P$  start from initial states  $s_0$  and  $s'_0$ , respectively, and if  $s$  and  $s'$  are the resulting final states, then  $s_0 = s'_0$  implies that `Post(s, s')` holds. The above rewritten specification suggests that we can relax the requirement of  $s_0 = s'_0$  by replacing it with a bridge predicate `Pre(s0, s'_0)`. For example:

```
deterministic assume(set.equals(set')) {
  set.add(3); || set.add(5);
} assert(set.equals(set'));
```

The above specification states that if any two executions start from sets containing the same elements, then after the execution of the code, the resulting sets must also contain the same elements.

**Comparison to Traditional Assertions:** In summary, we propose the following construct for the specification of deterministic behavior:

```
deterministic assume(Pre) {
  P
} assert(Post);
```

Formally, it states that for any two program states  $s_0$  and  $s'_0$ , if (1) `Pre(s0, s'_0)` holds, (2) an execution of  $P$  from  $s_0$  terminates and results in state  $s$ , and (3) an execution of  $P$  from  $s'_0$  terminates and results in state  $s'$ , then `Post(s, s')` must hold.

Note that the use of bridge predicates `Pre` and `Post` has the same flavor as pre- and postconditions used for functions in program verification. However, unlike traditional pre- and postconditions, the proposed `Pre` and `Post` predicates relate pairs of states from two different executions. In traditional verification, a precondition is usually written as a predicate over

a single program state, and a postcondition is usually written over two states—the states at the beginning and end of the function. For example:

```
parallel_matrix_multiply_int(A, B) {
  assume(A.cols == B.rows);
  ...
  assert(C == A × B);
  return C;
}
```

The key difference between a postcondition and a `Post` predicate is that a postcondition relates two states at different times along a same execution—e.g., here relating inputs `A` and `B` to output `C`—whereas a `Post` predicate relates two program states from different executions.

**Advantages of Deterministic Assertions:** Our deterministic specifications are a middle ground between the implicit specification used in race detection—that programs should be free of data races—and the full specification of functional correctness. It is a great feature of data race detectors that typically no programmer specification is needed. However, manually determining which reported races are benign and which are bugs can be time-consuming and difficult. We believe our deterministic assertions, while requiring little effort to write, can greatly aid in distinguishing harmful from benign data races (or higher-level races).

One could argue that a deterministic specification framework is unnecessary given that we can write the functional correctness of a block of code using traditional pre- and postconditions. For example, one could write the following to specify the correct behavior of a parallel matrix multiply:

```
C = parallel_matrix_multiply_float(A, B);
assert(|C - A × B| < 10-6);
```

We agree that if one can write a functional specification of a block of code, then there is no need to write deterministic specification, as functional correctness implies deterministic behavior.

The advantage of our deterministic assertions is that they provide a way to specify the correctness of just the use of parallelism in a program, independent of the program's full functional correctness. In many situations, writing a full specification of functional correctness is difficult and time-consuming. A simple deterministic specification, however, enables us to use automated techniques to check for parallelism bugs, such as harmful data races causing semantically nondeterministic behavior.

Consider a function `parallel_edge_detection` that takes an image as input and returns an image where detected edges have been marked. Relating the output to the input image with traditional pre- and postconditions would likely be quite challenging. However, it is simple to specify that the routine does not have any parallelism bugs causing a correct image to be returned for some thread schedules and an incorrect image for others:

```
deterministic assume(img.equals(img')) {
  result = parallel_edge_detection(img);
} assert(result.equals(result'));
```

where `img.equals(img')` returns true if the two images are pixel-by-pixel equal.

For this example, a programmer could gain some confidence in the correctness of the routine by writing unit tests or manually examining the output for a handful of images. He or she could then use automated testing or model checking to separately check that the parallel routine behaves deterministically on a variety of inputs, gaining confidence that the code is free from concurrency bugs.

We believe that it is often difficult to come up with effective functional correctness assertions. However, it is often quite easy to use bridge assertions to specify deterministic behavior, enabling a programmer to check for harmful concurrency bugs. In Section 5, we provide several case studies to support this argument.

### 3. CHECKING DETERMINISM

There may be many potential approaches to checking or verifying a deterministic specification, from testing to model checking to automated theorem proving. In this section, we propose a simple, sound, and incomplete method for checking deterministic specifications at run-time.

The key idea of the method is that, whenever a deterministic block is encountered at run-time, we can record the program states  $s_{pre}$  and  $s_{post}$  at the beginning and end of the block. Then, given a collection of  $(s_{pre}, s_{post})$  pairs for a particular deterministic block in some program, we can check a deterministic specification by comparing pairwise the pairs of initial and final states for the block. That is, for a deterministic block:

```
deterministic assume(Pre) {
  P
} assert(Post);
```

with pre- and postbridge predicates `Pre` and `Post`, we check for every recorded pair of pairs  $(s_{pre}, s_{post})$  and  $(s'_{pre}, s'_{post})$  that:

$$Pre(s_{pre}, s'_{pre}) \Rightarrow Post(s_{post}, s'_{post})$$

If this condition does not hold for some pair, then we report a determinism violation.

To increase the effectiveness of this checking, we must record pairs of initial and final states for deterministic blocks executed under a wide variety of possible thread interleavings and inputs. Thus, in practice we likely want to combine our deterministic assertion checking with existing techniques and tools for exploring parallel schedules of a program, such as noise making,<sup>7,18</sup> active random scheduling,<sup>16</sup> or model checking.<sup>20</sup>

In practice, the cost of recording and storing entire program states could be prohibitive. However, real determinism

predicates often depend on just a small portion of the whole program state. Thus, we need only to record and store small projections of program states. For example, for a deterministic specification with pre- and postpredicate `set.equals` (`set'`) we need only to save object `set` and its elements (possibly also the memory reachable from these objects), rather than the entire program memory. This storage cost sometimes can be further reduced by storing and comparing check-sums or approximate hashes.

#### 4. DETERMINISM CHECKING LIBRARY

In this section, we describe the design and implementation of an assertion library for specifying and checking determinism of Java programs. Note that, while it might be preferable to introduce a new syntactic construct for specifying determinism, we provide the functionality as a library to simplify the implementation.

##### 4.1. Overview

Figure 1 shows the core API for our deterministic assertion library. Functions `open` and `close` specify the beginning and end of a deterministic block. Deterministic blocks may be nested, and each block may contain multiple calls to functions `assume` and `assert`, which are used to specify the pre- and postpredicates of deterministic behavior.

Each call `assume(o, pre)` in a deterministic block specifies part of the prepredicate by giving some projection  $o$  of the program state and a predicate  $pre$ . That is, it specifies that one condition for any execution of the block to compute an equivalent, deterministic result is that `pre.apply(o, o')` return `true` for object  $o'$  from the other execution.

Similarly, a call `assert(o, post)` in a deterministic block specifies that, for any execution satisfying every `assume`, predicate `post.apply(o, o')` must return `true` for object  $o'$  from the other execution.

At run-time, our library records every object (i.e., state projection) passed to each `assert` and `assume` in each deterministic block, saving them to a central, persistent store. We require that all objects passed as state projections implement the `Serializable` interface to facilitate this recording. (In practice, this does not seem to be a heavy burden. Most core objects in the Java standard library are serializable, including numbers, strings, arrays, lists, sets, and maps/hashtables.)

**Figure 1. Core deterministic specification API.**

```
public class Deterministic {
    static void open() {...}
    static void close() {...}
    static void assume(Object o, Predicate p) {...}
    static void assert(Object o, Predicate p) {...}
    interface Predicate {
        boolean apply(Object a, Object b);
    }
}
```

Then, also at run-time, a call to `assert(o, post)` checks `post` on  $o$  and all  $o'$  saved from previous, matching executions of the same deterministic block. If the postpredicate does not hold for any of these executions, a determinism violation is immediately reported. Deterministic blocks can contain many `assert`'s so that determinism bugs can be caught as early as possible and can be more easily localized.

For flexibility, programmers are free to write state projections and predicates using the full Java language. However, it is a programmer's responsibility to ensure that these predicates contain no observable side effects, as there are no guarantees as to how many times such a predicate may be evaluated in any particular run.

**Built-in Predicates:** For programmer convenience, we provide two built-in predicates that are often sufficient for specifying pre- and postpredicates for determinism. The first, `Equals`, returns `true` if the given objects are equal using their built-in `equals` method—i.e., if `o.equals(o')`. For many Java objects, this method checks semantic equality—e.g., for integers, floating-point numbers, strings, lists, sets, etc. Further, for single- or multidimensional arrays (which do not implement such an `equals` method), the `Equals` predicate compares corresponding elements using their `equals` methods. Figure 2 gives an example with `assert` and `assume` using this `Equals` predicate.

The second predicate, `ApproxEquals`, checks if two floating-point numbers, or the corresponding elements of two floating-point arrays, are within a given margin of each other. We found this predicate useful in specifying the deterministic behavior of numerical applications, where it is often unavoidable that the low-order bits may vary with different thread interleavings.

**Real-World Floating-Point Predicates:** In practice, floating-point computations often have input-dependent error bounds. For example, we may expect any two runs of a parallel algorithm for summing inputs  $x_1, \dots, x_n$  to return answers

**Figure 2. Deterministic assertions for a Mandelbrot Set implementation from the Parallel Java (PJ) Library.<sup>11</sup>**

```
main(String args[]) {
    // Read parameters from command-line.
    ...
    // Pre-predicate: equal parameters.
    Predicate equals = new Equals();
    Deterministic.open();
    Deterministic.assume(width, equals);
    Deterministic.assume(height, equals);
    ...
    Deterministic.assume(gamma, equals);
    // spawn threads to compute fractal
    int matrix[][] = ...;
    ...
    // join threads
    ...
    Deterministic.assert(matrix, equals);
    Deterministic.close();

    // write fractal image to file
    ...
}
```



equal to within  $2N\epsilon\sum_i|x_i|$ , where  $\epsilon$  is the machine epsilon. We can assert:

```
sum = parallel_sum(x);
bound = 2 * x.length *  $\epsilon$  * sum_of_abs(x);
Predicate apx = new ApproxEquals(bound);
Deterministic.assert(sum, apx);
```

As another example, different runs of a molecular dynamics simulation may be expected to produce particle positions equal to within something like  $\epsilon$  multiplied by the sum of the absolute values of all initial positions. We can similarly compute this value at the beginning of the computation, and use an `ApproxEquals` predicate with the appropriate bound to compare particle positions.

#### 4.2. Concrete example: Mandelbrot

Figure 2 shows the deterministic assertions we added to one of our benchmarks, a program for rendering images of the Mandelbrot Set fractal from the Parallel Java (PJ) Library.<sup>11</sup>

The benchmark first reads a number of integer and floating-point parameters from the command-line. It then spawns several worker threads that each compute the hues for different segments of the final image and store the hues in shared array `matrix`. After waiting for all of the worker threads to finish, the program encodes and writes the image to a file given as a command-line argument.

To add determinism annotations to this program, we simply opened a deterministic block just before the worker threads are spawned and closed it just after they are joined. At the beginning of this block, we added an `assume` call for each of the seven fractal parameters, such as the image size and color palette. At the end of the block, we assert that the resulting array `matrix` should be deterministic, however the worker threads are interleaved.

Note that it would be quite difficult to add assertions for the functional correctness of this benchmark, as each

pixel of the resulting image is a complicated function of the inputs (i.e., the rate at which a particular complex sequence diverges). Further, there do not seem to be any simple traditional invariants on the program state or outputs which would help identify a parallelism bug.

## 5. EVALUATION

In this section, we describe our efforts to validate two claims about our proposal for specifying and checking deterministic parallel program execution:

1. First, deterministic specifications are easy to write. That is, even for programs for which it is difficult to specify traditional invariants or functional correctness, it is relatively easy for a programmer to add deterministic assertions.
2. Second, deterministic specifications are useful. When combined with tools for exploring multiple thread schedules, deterministic assertions catch real parallelism bugs that lead to semantic nondeterminism. Further, for traditional concurrency issues such as data races, these assertions provide some ability to distinguish between benign cases and true bugs.

To evaluate these claims, we used a number of benchmark programs from the Java Grande Forum (JGF) benchmark suite,<sup>17</sup> the Parallel Java (PJ) Library,<sup>11</sup> and elsewhere. The names and sizes of these benchmarks are given in Table 1. We describe the benchmarks in greater detail in Burnim and Sen.<sup>5</sup> Note that the benchmarks range from a few hundred to a few thousand lines of code, with the PJ benchmarks relying on an additional 10–20,000 lines of library code from the PJ Library (for threading, synchronization, and other functionality).

### 5.1. Ease of use

We evaluate the ease of use of our deterministic specification by manually adding assertions to our benchmark programs. One deterministic block was added to each benchmark.

**Table 1. Summary of experimental evaluation of deterministic assertions. A single deterministic block specification was added to each benchmark. Each specification was checked on executions with races found by the CALFUZZER<sup>14,16</sup> tool.**

Benchmark	Approximate Lines of Code (App + Library)	Lines of Specification (+ Predicates)	Threads	Data Races		High-Level Races	
				Found	Determinism Violations	Found	Determinism Violations
JGF							
sor	300	6	10	2	0	0	0
sparsematmult	700	7	10	0	0	0	0
series	800	4	10	0	0	0	0
crypt	1,100	5	10	0	0	0	0
moldyn	1,300	6	10	2	0	0	0
lufact	1,500	9	10	1	0	0	0
raytracer	1,900	4	10	3	<b>1</b>	0	0
montecarlo	3,600	4 + 34	10	1	0	2	0
PJ							
pi	150 + 15,000	5	4	9	0	1+	<b>1</b>
keysearch3	200 + 15,000	6	4	3	0	0+	0
mandelbrot	250 + 15,000	10	4	9	0	0+	0
phylogeny	4,400 + 15,000	8	4	4	0	0+	0
tsp	700	4	5	6	0	2	0

The third column of Table 1 records the number of lines of specification (and lines of custom predicate code) added to each benchmark. Overall, the specification burden is quite small. Indeed, for the majority of the programs, an author was able to add deterministic assertions in only 5 to 10 minutes per benchmark, despite being unfamiliar with the code. In particular, it was typically not difficult to both identify regions of code performing parallel computation and to determine from documentation, comments, or source code which results were intended to be deterministic. Figure 2 shows the assertions added to the `mandelbrot` benchmark.

The added assertions were correct on the first attempt for all but two benchmarks. For `phylogeny`, the resulting phylogenetic tree was erroneously specified as deterministic, when, in fact, there are many correct optimal trees. The specification was modified to assert only that the optimal score must be deterministic. For `sparsematmult`, we incorrectly identified the variable to which the output was written. This error was identified during later work on automatically inferring deterministic specifications.<sup>6</sup>

The two predicates provided by our assertion library were sufficient for all but one of the benchmarks. For the JGF `montecarlo` benchmark, the authors had to write a custom `equals` and `hashCode` method for two classes—34 total lines of code—in order to assume and assert that two sets, one of initial tasks and one of results, are equivalent across executions.

**Discussion:** More experience, or possibly user studies, would be needed to conclude decisively that our assertions are easier to use than existing techniques for specifying that parallel code is correctly deterministic. However, we believe our experience is quite promising. In particular, writing assertions for the full functional correctness of the parallel regions of these programs seemed to be quite difficult, perhaps requiring implementing a sequential version of the code and asserting that it produces the same result. Further, there seemed to be no obvious simpler, traditional assertions that would aid in catching nondeterministic parallelism.

Despite these difficulties, we found that specifying the natural deterministic behavior of the benchmarks with our bridge assertions required little effort.

## 5.2. Effectiveness

To evaluate the utility of our deterministic specifications in finding true parallelism bugs, we used a modified version of the CALFUZZER<sup>14, 16</sup> tool to find real races in the benchmark programs, both data races and higher level races (such as races to acquire a lock). For each such race, we ran 10 trials using CALFUZZER to create real executions with these races and to randomly resolve the races (i.e., randomly pick a thread to “win”). We turned on run-time checking of our deterministic assertions for these trials, and recorded all found violations.

Table 1 summarizes the results of these experiments. For each benchmark, we indicate the number of real data races and higher-level races we observed. Further, we indicate how many of these races led to determinism violations in any execution.

In these experiments, the primary computational cost is from CALFUZZER, which typically has an overhead in the range of 2x–20x for these kinds of compute bound applications. We have not carefully measured the computational cost of our deterministic assertion library. For most benchmarks, however, the cost of serializing and comparing a computation’s inputs and outputs is dwarfed by the cost of the computation itself—e.g., consider the cost of checking that two fractal images are identical versus the cost of computing each fractal in the first place.

**Determinism Violations:** We found two cases of nondeterministic behavior. First, a known data race in the `raytracer` benchmark, due the use of the wrong lock to protect a shared sum, can yield an incorrect final answer.

Second, the `pi` benchmark can yield a nondeterministic answer given the same random seed because of insufficient synchronization of a shared random number generator. In each Monte Carlo sample, two successive calls to `java.util.Random.nextDouble()` are made. A context switch between these calls changes the set of samples generated. Similarly, `nextDouble()` itself makes two calls to `java.util.Random.next()`, which atomically generates up to 32 pseudorandom bits. A context switch between these two calls changes the generated sequence of pseudorandom doubles. Thus, although `java.util.Random.nextDouble()` is thread-safe and free of data races, scheduling nondeterminism can still lead to a nondeterministic result. (This behavior is known—the PJ Library provides several versions of this benchmark, one of which does guarantee a deterministic result for any given random seed.)

**Benign Races:** The high number of real data races for these benchmarks is largely due to benign races on volatile variables used for synchronization—e.g., to implement a tournament barrier or a custom lock. Although CALFUZZER does not understand these sophisticated synchronization schemes, our deterministic assertions automatically provide some confidence that these races are benign because, over the course of many experimental runs, they did not lead to nondeterministic final results.

Note that it can be quite challenging to verify by hand that these races are benign. On inspecting the benchmark code and these data races, an author several times believed he had found a synchronization bug. But on deeper inspection, the code was found to be correct in all such cases.

The number of high-level races is low for the JGF benchmarks because all the benchmarks except `montecarlo` exclusively use volatile variables (and thread joins) for synchronization. Thus, all observable scheduling nondeterminism is due to data races.

The number of high-level races is low for the PJ benchmarks because they primarily use a combination of volatile variables and atomic compare-and-set operations for synchronization. Currently, the only kind of high-level race our modified CALFUZZER recognizes is a lock race. Thus, while we believe there are many (benign) races in the ordering of these compare-and-set operations, CALFUZZER does not report them. The one high-level race for `pi`, indicated in the table and described above, was confirmed by hand.

**Discussion:** Although our checking of deterministic assertions is sound—an assertion failure always indicates that two executions with equivalent initial states can yield nonequivalent final states—it is incomplete. Parallelism bugs leading to nondeterminism may still exist even when testing fails to find any determinism violations.

However, in our experiments we successfully distinguished the races known to cause undesired nondeterminism from the benign races in only a small number of trials. Thus, we believe our deterministic assertions can help catch harmful nondeterminism due to parallelism, as well as save programmer effort in determining whether real races and other potential parallelism bugs can lead to incorrect program behavior.

## 6. DISCUSSION

In this section, we compare the concepts of atomicity and determinism. Further, we discuss several other possible uses for bridge predicates and bridge assertions.

### 6.1. Atomicity versus determinism

A concept complementary to determinism in parallel programs is atomicity. A block of sequential code in a multithreaded program is said to be *atomic*<sup>9</sup> if for every possible interleaved execution of the program there exists an equivalent execution with the same overall behavior in which the atomic block is executed serially (i.e., the execution of the atomic block is not interleaved with actions of other threads). Therefore, if a code block is atomic, the programmer can assume that the execution of the code block by a thread cannot be interfered with by any other thread. This enables programmers to reason about atomic code blocks sequentially. This seemingly similar concept has the following subtle differences from determinism:

1. Atomicity is the property about a sequential block of code—i.e., the block of code for which we assert atomicity has a single thread of execution and does not spawn other threads. Note that a sequential block is by default deterministic if it is not interfered with by other threads. Determinism is a property of a parallel block of code. In determinism, we assume that the parallel block of code's execution is not influenced by the external world.
2. In atomicity, we say that the execution of a sequential block of code results in the same state no matter how it is scheduled with other external threads—i.e., atomicity ensures that *external nondeterminism* does not interfere with the execution of an atomic block of code. In determinism, we say that the execution of a parallel block of code gives the same semantic state no matter how the threads inside the block are scheduled—i.e., determinism ensures that *internal nondeterminism* does not result in different outputs.

In summary, *atomicity* and *determinism* are orthogonal concepts. Atomicity reasons about a single thread under external nondeterminism, whereas determinism reasons about multiple threads under internal nondeterminism.

Here we focus on atomicity and determinism as program specifications to be checked. There is much work on atomicity as a language mechanism, in which an atomic specification is instead *enforced* by some combination of library, run-time, compiler, or hardware support. One could similarly imagine enforcing deterministic specifications through, e.g., compiler and run-time mechanisms.<sup>4</sup>

### 6.2. Other uses of bridge predicates

We have already argued that bridge predicates simplify the task of directly and precisely specifying deterministic behavior of parallel programs. We also believe that bridge predicates could provide a simple but powerful tool to express correctness properties in many other situations. For example, if we have two versions of a program, P1 and P2, that we expect to produce the same output on the same input, then we can easily assert this using our framework as follows:

```
deterministic assume (Pre) {
    if (nonDeterministicBoolean()) {
        P1
    } else {
        P2
    }
} assert (Post);
```

where `Pre` requires that the inputs are the same and `Post` specifies that the outputs will be the same.

In particular, if a programmer has written both a sequential and parallel version of a piece of code, he or she can specify that the two versions are semantically equivalent with an assertion like:

```
deterministic assume (A==A' and B==B') {
    if (nonDeterministicBoolean()) {
        C = par_matrix_multiply_int(A, B);
    } else {
        C = seq_matrix_multiply_int(A, B);
    }
} assert (C==C');
```

where `nonDeterministicBoolean()` returns true or false nondeterministically.

Similarly, a programmer can specify that the old and new versions of a piece of code are semantically equivalent:

```
deterministic assume (A==A' and B==B') {
    if (nonDeterministicBoolean()) {
        C = old_matrix_multiply_int(A, B);
    } else {
        C = new_matrix_multiply_int(A, B);
    }
} assert (C==C');
```



Checking this specification is a kind of regression testing. In particular, if the code change has introduced a regression—i.e., a bug that causes the new code to produce a semantically different output than the old code for some input—then the above specification does not hold.

Further, we believe there is a wider class of program properties that are easy to write in bridge assertions but would be quite difficult to write otherwise. For example, consider the specification:

```
deterministic assume(set.size() == set'.size()) {
    P
} assert (set.size() == set'.size());
```

This specification requires that sequential or parallel program block *P* transforms *set* so that its final size is some function of its initial size, independent of its elements. The specification is easy to write even in cases where the exact relationship between the initial and final size might be quite complex and difficult to write. It is not entirely clear, however, when such properties are important or useful to specify.

## 7. CONCLUSION

We have proposed bridge predicates and bridge assertions for specifying the user-intended semantic deterministic behavior of parallel programs. We argue that our specifications are much simpler for programmers to write than traditional specifications of functional correctness, because they enable programmers to compare pairs of program states across different executions rather than relating program outputs directly to program inputs. Thus, bridge predicates and bridge assertions can be thought of as a lightweight mechanism for specifying the correctness of just the parallelism in a program, independently of the program's functional correctness.

We have shown experimental evidence that we can effectively check our deterministic specifications. In particular, we can use existing techniques for testing parallel software to generate executions exhibiting data and higher-level races. Then our deterministic specifications allow us to distinguish from the benign races the parallel nondeterministic bugs that lead to unintended nondeterministic program behavior. Thus, we argue that it is worthwhile for programmers to write such lightweight deterministic specifications. In fact, later work<sup>6</sup> has suggested that, given the simple form of our specifications, it may often be possible to automatically infer likely deterministic specifications for parallel programs.

## Acknowledgments

We would like to thank Nicholas Jalbert, Mayur Naik, Chang-Seo Park, and our anonymous reviewers for their valuable comments on previous drafts of this paper. This work supported in part by Microsoft (Award #024263) and

Intel (Award #024894) funding and by matching funding by U.C. Discovery (Award #DIG07-10227), by NSF Grants CNS-0720906 and CCF-0747390, and by a DoD NDSEG Graduate Fellowship. □

## References

1. Artho, C., Havelund, K., Biere, A. High-level data races. *Softw. Test. Ver. Reliab.* 13, 4 (2003), 207–227.
2. Asanovic, K., Bodik, R., Demmel, J., Keaveny, T., Keutzer, K., Kubitowicz, J.D., Lee, E.A., Morgan, N., Necula, G., Patterson, D.A., Sen, K., Wawrzynek, J., Wessel, D., Yelick, K.A. *The Parallel Computing Laboratory at U.C. Berkeley: A Research Agenda Based on the Berkeley View*. Technical Report UCB/EECS-2008-23, EECS Department, University of California, Berkeley, March 2008.
3. Barnes, G. A method for implementing lock-free shared-data structures. In *5th ACM Symposium on Parallel Algorithms and Architectures (SPAA)* (1993).
4. Bocchino, R.L., Jr., Adve, V.S., Dig, D., Adve, S.V., Heumann, S., Komuravelli, R., Overbey, J., Simmons, P., Sung, H., Vakilian, M. A type and effect system for deterministic parallel Java. In *24th ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA)* (2009).
5. Burnim, J., Sen, K. Asserting and checking determinism for multithreaded programs. In *7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE)* (2009).
6. Burnim, J., Sen, K. DETERMIN: Inferring likely deterministic specifications of multithreaded programs. In *32nd ACM/IEEE International Conference on Software Engineering (ICSE)* (2010).
7. Edelstein, O., Farchi, E., Nir, Y., Ratsaby, G., Ur, S. Multithreaded Java program test generation. *IBM Syst. J.* 41, 1 (2002), 111–125.
8. Flanagan, C., Freund, S.N. Atomizer: A dynamic atomicity checker for multithreaded programs. In *31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)* (2004).
9. Flanagan, C., Qadeer, S. A type and effect system for atomicity. In *ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation (PLDI)* (2003).
10. Johnston, W.M., Hanna, J.R.P., Millar, R.J. Advances in dataflow programming languages. *ACM Comput. Surv.* 36, 1 (2004), 1–34.
11. Kaminsky, A. Parallel Java: A unified API for shared memory and cluster parallel programming in 100% Java. In *21st IEEE International Parallel and Distributed Processing Symposium (IPDPS)* (2007).
12. Lee, E.A. The problem with threads. *Computer* 39, 5 (May 2006), 33–42.
13. Loidl, H., Rubio, F., Scaife, N., Hammond, K., Horiguchi, S., Klusik, U., Loogen, R., Michaelson, G., Pena, R., Priebe, S. et al. Comparing parallel functional languages: Programming and performance. *High. Order Symb. Comput.* 16, 3 (2003), 203–251.
14. Park, C.-S., Sen, K. Randomized active atomicity violation detection in concurrent programs. In *16th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE)* (2008).
15. Sadowski, C., Freund, S., Flanagan, C. SingleTrack: A dynamic determinism checker for multithreaded programs. In *18th European Symposium on Programming (ESOP)* (2009).
16. Sen, K. Race directed random testing of concurrent programs. In *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'08)* (2008).
17. Smith, L.A., Bull, J.M., Obdržálek, J. A parallel java grande benchmark suite. In *ACM/IEEE Conference on Supercomputing (SC)* (2001).
18. Stoller, S.D. Testing concurrent Java programs using randomized scheduling. In *2nd Workshop on Runtime Verification (RV)* (2002).
19. Thies, W., Karczmarek, M., Amarasinghe, S. StreamIt: A language for streaming applications. In *11th International Conference on Compiler Construction (CC)* (2002).
20. Visser, W., Havelund, K., Brat, G., Park, S., Lerda, F. Model checking programs. *Autom. Softw. Eng.* 10, 2 (2003), 203–232.
21. von Praun, C., Gross, T.R. Object race detection. In *16th ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)* (2001).

Jacob Burnim (jburnim@cs.berkeley.edu),  
EECS Department, UC Berkeley, CA.

Koushik Sen (ksen@cs.berkeley.edu),  
EECS Department, UC Berkeley, CA.

# Technical Perspective

## Learning To Do Program Verification

By K. Rustan M. Leino

WHEN YOU DECIDE to use a piece of software, how do you know it will do what you need it to do or what it claims to do? Will it even be safe to run? Will it interfere with other software you already have?

As a software engineer developing that piece of software, what can you do to ensure the software will be ready in time and will meet your quality standards?

A long-standing dream of ideal software development calls for the use of program verification as part of the answers to these questions. Program verification is the process by which one develops a mathematical proof that shows the software satisfies its functional specification. This dream had grown roots already by the 1970s.

Since then, and especially in the last decade, the technology behind program verification has become an important part of software development practice. At Microsoft alone, a number of symbolic execution techniques, including abstract interpretation, counterexample-guided predicate abstraction, and satisfiability-modulo-theories solving, are used routinely to enhance the testing of software, in some cases even mathematically verifying the absence of certain kinds of software defects.

But what we use today pales in comparison to the grand dreams of the 1970s. Why? Is it not possible to provide rigorous proofs of modern software? Is it too difficult to capture in functional specifications the intended behavior of software? Is full program verification only for small algorithms and toy examples? Are hardcore programmers and verification experts not able to find common goals to work toward? Is it not cost-effective to insist that every detail of a piece of software is correct?

The following work by Gerwin Klein et al. is a landmark in the further development of applying functional-correctness verification in practice. The authors have produced a machine-

checkable proof of correctness for the microkernel of an operating system. For a user of the operating system, this verification provides a number of significant benefits; for example, the assurance that no hostile application running under this operating system can subvert the integrity of the kernel through a buffer-overflow attack. For the software engineers involved, the verification brings the benefit of a code analysis that far surpasses that achieved by testing alone.

The seL4 verification connects three descriptions of the system: an abstract specification of the kernel that describes what its functional behavior is, a high-level prototype implementation of the system that introduces further details of how the kernel performs its operations, and a low-level, hand-optimized implementation that deals with the nitty-gritty of the kernel's operation. A novel aspect of this project is how the middle layer was used as a stepping-stone not just for the verification, but also for the actual design and implementation of the system itself. This has the great advantage that the verification process gets the chance to influence the design. As the paper reports, this led to a large number of changes in the top two layers, with the effect of boosting the productivity of both the design team and the verification team.

**A project like seL4 verification is not easy to pull off. In short, it is a big bet. I applaud the team for undertaking it.**

The authors are careful to list not only positive implications of the verification, but also the assumptions upon which the verification rests. This is important, for all verification is performed at some level of abstraction. For example, verifying a program at the level of abstraction provided by a programming language does not say anything about hardware failures. Verification is not an absolute; what it seeks to do is offer detailed aid for programmers at the level of abstraction at which they are working.

A project like the seL4 verification is not easy to pull off. It takes a strong vision, as well as a significant amount of work, expertise, and persistence. In short, it is a big bet. I applaud the team for undertaking it, and congratulate them on delivering. Any doubts as to the technical feasibility of such a project should by now have been removed.

The question about cost-effectiveness, however, remains. Some may argue that the end result of the verification—a level of assurance that could not have been obtained using more traditional methods—has already made up for the effort expended. Others may balk at the 20 person-years to complete the proof or at the ratio of 200,000 lines of proof script to 6,000 lines of eventual C code. I would like to offer a different perspective on these numbers. First, they provide a benchmark against which to compare future work. I would expect that in another decade, a similar project will take less effort and will involve a larger degree of automation. Second, the effort has resulted not just in an impressive engineering achievement, but also in an appreciable amount of scientific learning. It is through pioneering and repeated efforts like this one that we will learn how to apply full program verification on a more regular basis. ■

K. Rustan M. Leino (leino@microsoft.com) is a Principal Researcher at Microsoft Research, Redmond, WA.

© 2010 ACM 0001-0782/10/0600 \$10.00

# seL4: Formal Verification of an Operating-System Kernel

By Gerwin Klein, June Andronick, Kevin Elphinstone, Gernot Heiser, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood

## Abstract

**We report on the formal, machine-checked verification of the seL4 microkernel from an abstract specification down to its C implementation. We assume correctness of compiler, assembly code, hardware, and boot code.**

**seL4 is a third-generation microkernel of L4 provenance, comprising 8700 lines of C and 600 lines of assembler. Its performance is comparable to other high-performance L4 kernels.**

**We prove that the implementation always strictly follows our high-level abstract specification of kernel behavior. This encompasses traditional design and implementation safety properties such as that the kernel will never crash, and it will never perform an unsafe operation. It also implies much more: we can predict precisely how the kernel will behave in every possible situation.**

## 1. INTRODUCTION

Almost every paper on formal verification starts with the observation that software complexity is increasing, that this leads to errors, and that this is a problem for mission and safety critical software. We agree, as do most.

Here, we report on the full formal verification of a critical system from a high-level model down to very low-level C code. We do not pretend that this solves all of the software complexity or error problems. We do think that our approach will work for similar systems. The main message we wish to convey is that a formally verified commercial-grade, general-purpose microkernel now exists, and that formal verification is possible and feasible on code sizes of about 10,000 lines of C. It is not cheap; we spent significant effort on the verification, but it appears cost-effective and more affordable than other methods that achieve lower degrees of trustworthiness.

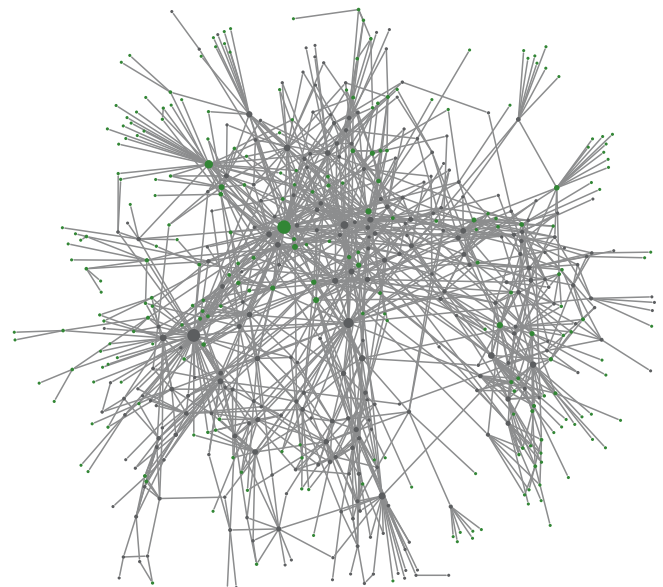
To build a truly trustworthy system, one needs to start at the operating system (OS) and the most critical part of the OS is its *kernel*. The kernel is defined as the software that executes in the privileged mode of the hardware, meaning that there can be no protection from faults occurring in the kernel, and every single bug can potentially cause arbitrary damage. The kernel is a mandatory part of a system's *trusted computing base* (TCB)—the part of the system that can bypass security.<sup>10</sup> Minimizing this TCB is the core concept behind *microkernels*, an idea that goes back 40 years.

A microkernel, as opposed to the more traditional *monolithic* design of contemporary mainstream OS kernels, is reduced to just the bare minimum of code wrapping

hardware mechanisms and needing to run in privileged mode. All OS services are then implemented as normal programs, running entirely in (unprivileged) user mode, and therefore can potentially be excluded from the TCB. Previous implementations of microkernels resulted in communication overheads that made them unattractive compared to monolithic kernels. Modern design and implementation techniques have managed to reduced this overhead to very competitive limits.

A microkernel makes the trustworthiness problem more tractable. A well-designed high-performance microkernel, such as the various representatives of the L4 microkernel family, consists of the order of 10,000 lines of code (10 kloc). This radical reduction to a bare minimum comes with a price in complexity. It results in a high degree of interdependency between different parts of the kernel, as indicated in Figure 1. Despite this increased complexity in low-level code, we have demonstrated that with modern techniques and careful design, an OS microkernel is entirely within the realm of full formal verification.

**Figure 1. Call graph of the seL4 microkernel. Vertices represent functions, and edges invocations.**



The original version of this paper was published in the *Proceedings of the 22nd ACM SIGOPS Symposium on Operating Systems Principles*, Oct. 2009.



Formal verification of software refers to the application of mathematical proof techniques to establish properties about programs. Formal verification can cover not just all lines of code or all decisions in a program, but all possible behaviors for all possible inputs. For example, the very simple fragment of C code `if (x < y) z = x/y else z = y/x` for `x`, `y`, and `z` being `int` tested with `x = 4`, `y = 2` and `x = 8`, `y = 16`, results in full code coverage: every line is executed at least once, and every branch of every condition is taken at least once. Yet, there are still two potential bugs remaining. Of course, any human tester will find inputs such as `x = 0`, `y = -1` and `x = -1`, `y = 0` that expose the bugs, but for bigger programs it is infeasible to be sure of completeness. This is what formal verification can achieve.

The approach we use is interactive, machine-assisted, and machine-checked proof. Specifically, we use the theorem prover Isabelle/HOL.<sup>8</sup> Interactive theorem proving requires human intervention and creativity to construct and guide the proof. It has the advantage that it is not constrained to specific properties or finite, feasible state spaces. We have proved the *functional correctness* of the seL4 microkernel, a secure embedded microkernel of the L4<sup>6</sup> family. This means, we have proved mathematically that the implementation of seL4 always strictly follows our high-level abstract specification of kernel behavior. This property is stronger and more precise than what automated techniques like model checking, static analysis, or kernel implementations in type-safe languages can achieve. We not only analyze specific aspects of the kernel, such as safe execution, but also provide a full specification and proof for the kernel's precise behavior.

In the following, we describe what the implications of the proof are, how the kernel was designed for verification, what the verification itself entailed and what its assumptions are, and finally what effort it cost us.

## 2. IMPLICATIONS

In a sense, functional correctness is one of the strongest properties to prove about a system. Once we have proved functional correctness with respect to a model, we can use this model to establish further properties instead of having to reason directly about the code. For instance, we prove that every system call terminates by looking at the model instead of the code. However, there are some security-relevant properties, such as transmission of information via covert channels, for which the model may not be precise enough.

So our proof does not mean that seL4 is secure for all purposes. We proved that seL4 is functionally correct. *Secure* would first need a formal definition and depends on the application. Taken seriously, security is a whole-system question, including the system's human components.

Even without proving specific security properties on top, a functional correctness proof already has interesting implications for security. If the assumptions listed in Section 4.5 are true, then in seL4 there will be:

**No code injection attacks:** If we always know precisely what the system does and if the spec does not explicitly allow it, then we can never have any foreign code executing as part of seL4.

**No buffer overflows:** This is mainly a classic vector for code injection, but buffer overflows may also inject unwanted data and influence kernel behavior that way. We prove that all array accesses are within bounds and we prove that all pointer accesses are well typed, even if they go via casts to void or address arithmetic.

**No NULL pointer access:** NULL pointer bugs can allow local privilege escalation and execution of arbitrary code in kernel mode.<sup>9</sup> Absence of NULL pointer dereference is a direct proof obligation for us for every pointer access.

**No ill-typed pointer access:** Even though the kernel code deliberately breaks C type safety for efficiency at some points, in order to predict that the system behaves according to specification, we prove that circumventing the type system is safe at all these points.

**No memory leaks** and no memory freed that is still in use. This is not purely a consequence of the proof itself. Much of the design of seL4 was focused on explicit memory management. Users may run out of memory, but the kernel never will.

**No nontermination:** We have proved that all kernel calls terminate. This means the kernel will never suddenly freeze and not return from a system call. This does not mean that the whole system will never freeze. It is still possible to write bad device drivers and bad applications, but set up correctly, a supervisor process can always stay in control of the rest of the system.

**No arithmetic or other exceptions:** The C standard defines a long list of things that can go wrong and that should be avoided: shifting machine words by a too-large amount, dividing by zero, etc. We proved explicitly that none of these occur, including the absence of errors due to overflows in integer arithmetic.

**No unchecked user arguments:** All user input is checked and validated. If the kernel receives garbage or malicious arguments it will respond with the specified error messages, not with crashes. Of course, the kernel will allow a thread to kill itself if that thread has sufficient capabilities. It will never allow anything to crash the kernel, though.

Many of these are general security traits that are good to have for any kind of system. We have also proved a large number of properties that are specific to seL4. We have proved them about the kernel design and specification. With functional correctness, we know they are true about the code as well. Some examples are:

**Aligned objects:** Two simple low-level invariants of the kernel are: all objects are aligned to their size, and no two objects overlap in memory. This makes comparing memory regions for objects very simple and efficient.

**Well-formed data structures:** Lists, doubly linked, singly linked, with and without additional information, are a pet topic of formal verification. These data structures also occur in seL4 and we proved the usual properties: lists are not circular when they should not be, back pointers point to the right nodes, insertion, deletion etc., work as expected.

**Algorithmic invariants:** Many optimizations rely on certain properties being always true, so specific checks can be left out or can be replaced by other, more efficient checks. A simple example is that the distinguished idle thread is always in thread state *idle* and therefore can never be blocked or

otherwise waiting for I/O. This can be used to remove checks in the code paths that deal with the idle thread.

**Correct book-keeping:** The seL4 kernel has an explicit user-visible concept of keeping track of memory, who has access to it, who access was delegated to, and what needs to be done if a privileged process wants to revoke access from delegates. It is the central mechanism for reusing memory in seL4. The data structure that backs this concept is correspondingly complex and its implications reach into almost all aspects of the kernel. For instance, we proved that if a live object exists anywhere in memory, then there exists an explicit capability node in this data structure that covers the object. And if such a capability exists, then it exists in the proper place in the data structure and has the right relationship towards parents, siblings, and descendants within. If an object is live (may be mentioned in other objects anywhere in the system) then the object itself together with that capability must have recorded enough information to reach all objects that refer to it (directly or indirectly). Together with a whole host of further invariants, these properties allow the kernel code to reduce the complex, system-global test whether a region of memory is mentioned anywhere else in the system to a quick, local pointer comparison.

We have proved about 80 such invariants on the executable specification such that they directly transfer to the data structures used in the C program.

A verification like this is not an absolute guarantee. The key condition in all this is *if the assumptions are true*. To attack any of these properties, this is where one would have to look. What the proof really does is take 7500 lines of C code out of the equation. It reduces possible attacks and the human analysis necessary to guard against them to the assumptions and specification. It also is the basis for any formal analysis of systems running on top of the kernel or for further high-level analysis of the kernel itself.

### 3. KERNEL DESIGN FOR VERIFICATION

The challenge in designing a verifiable and usable kernel lies in reducing complexity to make verification easier while maintaining high performance.

To achieve these two objectives, we designed and implemented a microkernel from scratch. This kernel, called seL4, is a third-generation microkernel, based on L4 and influenced by EROS.<sup>11</sup> It is designed for practical deployment in embedded systems with high trustworthiness requirements. One of its innovations is completely explicit memory-management subject to policies defined at user level, even for kernel memory. All authority in seL4 is mediated by *capabilities*,<sup>2</sup> tokens identifying objects and conveying access rights.

We first briefly present the approach we used for a kernel/proof codesign process. Then we highlight the main design decisions we made to simplify the verification work.

#### 3.1. Kernel/proof codesign process

One key idea in this project was bridging the gap between verifiability and performance by using an iterative approach to kernel design, based around an intermediate target that is readily accessible to both OS developers and formal methods practitioners. We used the functional language Haskell

to provide a programming language for OS developers, while at the same time providing an artifact that can readily be reasoned about in the theorem proving tool: the design team wrote increasingly complete prototypes of the kernel in Haskell, exporting the system call interface via a hardware simulator to user-level binary code. The formal methods team imported this prototype into the theorem prover and used it as an intermediate executable specification. The approach aims at quickly iterating through design, prototype implementation, and formal model until convergence.

Despite its ability to run real user code, the Haskell kernel remains a prototype, as it does not satisfy our high-performance requirement. Furthermore, Haskell requires a significant run-time environment (much bigger than our kernel), and thus violates our requirement of a small TCB. We therefore translated the Haskell implementation *manually* into high-performance C code. An automatic translation (without proof) would have been possible, but we would have lost most opportunities to micro-optimize the kernel in order to meet our performance targets. We do not need to trust the translations into C and from Haskell into Isabelle—we formally verify the C code as it is seen by the compiler, gaining an end-to-end theorem between formal specification and the C semantics.

#### 3.2. Design decisions

**Global Variables and Side Effects:** Use of global variables and functions with side effects is common in operating systems—mirroring properties of contemporary computer hardware and OS abstractions. Our verification techniques can deal routinely with side effects, but implicit state updates and complex use of the same global variable for different purposes make verification more difficult. This is not surprising: the higher the conceptual complexity, the higher the verification effort.

The deeper reason is that global variables usually require stating and proving invariant properties. For example, scheduler queues are global data structures frequently implemented as doubly linked lists. The corresponding invariant might state that all back links in the list point to the appropriate nodes and that all elements point to thread control blocks and that all active threads are in one of the scheduler queues.

Invariants are expensive because they need to be proved not only locally for the functions that directly manipulate the scheduler queue, but for the whole kernel—we have to show that no other pointer manipulation in the kernel destroys the list or its properties. This proof can be easy or hard, depending on how modularly the global variable is used.

Dealing with global variables was simplified by deriving the kernel implementation from Haskell, where side effects are explicit and drawn to the design team's attention.

**Kernel Memory Management:** The seL4 kernel uses a model of memory allocation that exports control of the in-kernel allocation to appropriately authorized applications. While this model is mostly motivated by the need for precise guarantees of memory consumption, it also benefits verification. The model pushes the policy for allocation outside the kernel, which means we only need to prove that the mechanism

works, not that the user-level policy makes sense. The mechanism works if it keeps kernel code and data structures safe from user access, if the virtual memory (VM) subsystem is fully controlled by the kernel interface via capabilities, and if it provides the necessary functionality for user level to manage its own VM policies.

Obviously, moving policy into user land does not change the fact that memory allocation is part of the TCB. It does mean, however, that memory allocation can be verified separately, and can rely on verified kernel properties.

The memory-management model gives free memory to the user-level manager in the form of regions tagged as *untyped*. The memory manager can split untyped regions and retype them into one of several kernel object types (one of them, *frame*, is for user-accessible memory); such operations create new capabilities. Object destruction converts a region back to untyped (and invalidates derived capabilities).

Before reusing a block of memory, all references to this memory must be invalidated. This involves either finding all outstanding capabilities to the object, or returning the object to the memory pool only when the last capability is deleted. Our kernel uses both approaches. In the first approach, a so-called capability derivation tree is used to find and invalidate all capabilities referring to a memory region. In the second approach, the capability derivation tree is used to ensure, with a check that is local in scope, that there are no system-wide dangling references. This is possible because all other kernel objects have further invariants on their own internal references that relate back to the existence of capabilities in this derivation tree.

Similar book-keeping would be necessary for a traditional *malloc/free* model in the kernel. The difference is that the complicated *free* case in our model is concentrated in one place, whereas otherwise it would be repeated numerous times over the code.

**Concurrency and Nondeterminism:** Concurrency is the execution of computation in parallel (in the case of multiple hardware processors), or by nondeterministic interleaving via a concurrency abstraction like threads. Reasoning about concurrent programs is hard, much harder than reasoning about sequential programs. For the time being, we limited the verification to a single-processor version of seL4.

In a uniprocessor kernel, concurrency can result from three sources: *yielding* of the processor from one thread to another, (synchronous) *exceptions* and (asynchronous) *interrupts*. Yielding can be synchronous, by an explicit handover, such as when blocking on a lock, or asynchronous, by preemption (but in a uniprocessor kernel, the latter can only happen as the result of an interrupt).

We limit the effect of all three by a kernel design which explicitly minimizes concurrency.

*Exceptions* are completely avoided, by ensuring that they never occur. For instance, we avoid virtual-memory exceptions by allocating all kernel data structures in a region of VM which is always guaranteed to be mapped to physical memory. System-call arguments are either passed in registers or through preregistered physical memory frames.

The complexity of synchronous *yield* we avoid by using an event-based kernel execution model, with a single kernel

stack, and a mostly atomic application programming interface. This is aided by the traditional L4 model of system calls which are primitive and mostly short-running.

We minimize the effect of interrupts (and hence preemptions) by disabling interrupts during kernel execution. Again, this is aided by the L4 model of short system calls.

However, not all kernel operations can be guaranteed to be short; object destruction especially can require almost arbitrary execution time, so not allowing any interrupt processing during a system call would rule out the use of the kernel for real-time applications, undermining the goal of real-world deployability.

We ensure bounded interrupt latencies by the standard approach of introducing a few, carefully placed, *interrupt points*. On detection of a pending interrupt, the kernel explicitly returns through the function call stack to the kernel/user boundary and responds to the interrupt. It then restarts the original operation, including reestablishing all the preconditions for execution. As a result, we completely avoid concurrent execution in the kernel.

**I/O:** Interrupts are used by *device drivers* to affect I/O. L4 kernels traditionally implement device drivers as user-level programs, and seL4 is no different. Device interrupts are converted into messages to the user-level driver.

This approach removes a large amount of complexity from the kernel implementation (and the proof). The only exception is an in-kernel timer driver which generates timer ticks for scheduling, which is straightforward to deal with.

#### 4. VERIFICATION OF seL4

This section gives an overview of the formal verification of seL4 in the theorem prover Isabelle/HOL.<sup>8</sup> The property we are proving is functional correctness. Formally, we are showing *refinement*: A refinement proof establishes a correspondence between a high-level (abstract) and a low-level (concrete, or *refined*) representation of a system.

The correspondence established by the refinement proof ensures that all Hoare logic properties of the abstract model also hold for the refined model. This means that if a security property is proved in Hoare logic about the abstract model (not all security properties can be), our refinement guarantees that the same property holds for the kernel source code. In this paper, we concentrate on the general functional correctness property. We have also modelled and proved the security of seL4's access-control system in Isabelle/HOL on a high level.<sup>3</sup>

Figure 2 shows the specification layers used in the verification of seL4; they are related by formal proof. In the following sections we explain each layer in turn.

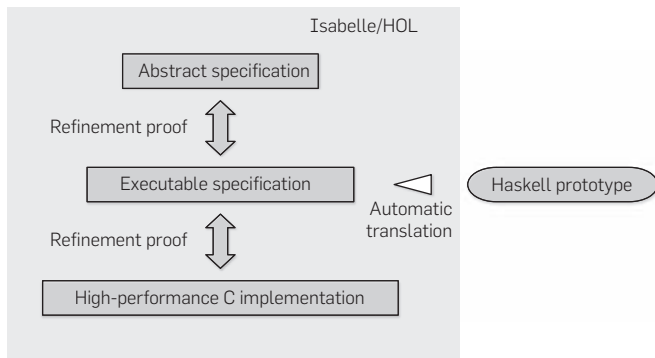
##### 4.1. Abstract specification

The abstract level describes *what* the system does without saying *how* it is done. For all user-visible kernel operations, it describes the functional behavior that is expected from the system. All implementations that refine this specification will be binary compatible.

We precisely describe argument formats, encodings and error reporting, so, for instance, some of the C-level size restrictions become visible on this level. We model finite machine words, memory, and typed pointers explicitly.



**Figure 2. The refinement layers in the verification of seL4.**



Otherwise, the data structures used in this abstract specification are high level—essentially sets, lists, trees, functions, and records. We make use of nondeterminism in order to leave implementation choices to lower levels: if there are multiple correct results for an operation, this abstract layer would return all of them and make clear that there is a choice. The implementation is free to pick any one of them.

An example of this is scheduling. No scheduling policy is defined at the abstract level. Instead, the scheduler is modelled as a function picking *any* runnable thread that is active in the system *or* the idle thread. The Isabelle/HOL code for this is shown in Figure 3. The function `all_active_tcbs` returns the abstract set of all runnable threads in the system. Its implementation (not shown) is an abstract logical predicate over the whole system. The `select` statement picks any element of the set. The `OR` makes a nondeterministic choice between the first block and `switch_to_idle_thread`. The executable specification makes this choice more specific.

#### 4.2. Executable specification

The purpose of the executable specification is to fill in the details left open at the abstract level and to specify how the kernel works (as opposed to what it does). While trying to avoid the messy specifics of how data structures and code are optimized in C, we reflect the fundamental restrictions in size and code structure that we expect from the hardware and the C implementation. For instance, we take care not to use more than 64 bits to represent capabilities, exploiting known alignment of pointers. We do not specify in which way this limited information is laid out in C.

The executable specification is deterministic; the only nondeterminism left is that of the underlying machine. All

**Figure 3. Isabelle/HOL code for scheduler at abstract level.**

```
schedule ≡ do
  threads ← all_active_tcbs;
  thread ← select threads;
  switch_to_thread thread
od OR switch_to_idle_thread
```

data structures are now explicit data types, records, and lists with straightforward, efficient implementations in C. For example the capability derivation tree of seL4, modelled as a tree on the abstract level, is now modelled as a doubly linked list with limited level information. It is manipulated explicitly with pointer-update operations.

Figure 4 shows part of the scheduler specification at this level. The additional complexity becomes apparent in the `chooseThread` function that is no longer merely a simple predicate, but rather an explicit search backed by data structures for priority queues. The specification fixes the behavior of the scheduler to a simple priority-based round-robin algorithm. It mentions that threads have time slices and it clarifies when the idle thread will be scheduled. Note that priority queues duplicate information that is already available (in the form of thread states), in order to make it available *efficiently*. They make it easy to find a runnable thread of high priority. The optimization will require us to prove that the duplicated information is consistent.

We have proved that the executable specification correctly implements the abstract specification. Because of its extreme level of detail, this proof alone already provides stronger design assurance than has been shown for any other general-purpose OS kernel.

#### 4.3. C implementation

The most detailed layer in our verification is the C implementation. The translation from C into Isabelle is correctness-critical and we take great care to model the semantics of our C subset precisely and foundationally. *Precisely* means that we treat C semantics, types, and memory model as the C99 standard<sup>4</sup> prescribes, for instance, with architecture-dependent word size, padding of structs, type-unsafe casting of pointers, and arithmetic on addresses. As kernel programmers do, we make assumptions about the compiler (GCC) that go beyond the standard, and about the architecture

**Figure 4. Haskell code for schedule.**

```
schedule = do
  action <- getSchedAction
  case action of
    ChooseNewThread -> do
      chooseThread
      setSchedAction ResumeCurrentThread
      ...
  chooseThread = do
    r <- findM chooseThread' (reverse [minBound ..
      maxBound])
    when (r == Nothing) $ switchToIdleThread
  chooseThread' prio = do
    q <- getQueue prio
    liftM isJust $ findM chooseThread" q
  chooseThread" thread = do
    runnable <- isRunnable thread
    if not runnable then do
      tcbSchedDequeue thread
      return False
    else do
      switchToThread thread
      return True
```

used (ARMv6). These are explicit in the model, and we can therefore detect violations. *Foundationally* means that we do not just axiomatize the behavior of C on a high level, but we derive it from first principles as far as possible. For example, in our model of C, memory is a primitive function from addresses to bytes without type information or restrictions. On top of that, we specify how types like unsigned int are encoded, how structures are laid out, and how implicit and explicit type casts behave. We managed to lift this low-level memory model to a high-level calculus that allows efficient, abstract reasoning on the type-safe fragment of the kernel. We generate proof obligations assuring the safety of each pointer access and write. They state that the pointer in question must be non-null and of the correct alignment. They are typically easy to discharge. We generate similar obligations for all restrictions the C99 standard demands.

We treat a very large, pragmatic subset of C99 in the verification. It is a compromise between verification convenience and the hoops the kernel programmers were willing to jump through in writing their source. The following paragraphs describe what is *not* in this subset.

We do not allow the address-of operator & on local variables, because, for better automation, we make the assumption that local variables are separate from the heap. This could be violated if their address was available to pass on. It is the most far-reaching restriction we implement, because it is common in C to use local variable references for return parameters to avoid returning large types on the stack. We achieved compliance with this requirement by avoiding reference parameters as much as possible, and where they were needed, used pointers to global variables (which are not restricted).

One feature of C that is problematic for verification (and programmers) is the unspecified order of evaluation in expressions with side effects. To deal with this feature soundly, we limit how side effects can occur in expressions. If more than one function call occurs within an expression or the expression otherwise accesses global state, a proof obligation is generated to show that these functions are side-effect free. This proof obligation is discharged automatically.

We do not allow function calls through function pointers. (We do allow handing the address of a function to assembler code, e.g., for installing exception vector tables.) We also do not allow goto statements, or switch statements with fall-through cases. We support C99 compound literals, making it convenient to return structs from functions, and reducing the need for reference parameters. We do not allow compound literals to be *lvalues*. Some of these restrictions could be lifted easily, but the features were not required in seL4.

We did not use unions directly in seL4 and therefore do not support them in the verification (although that would be possible). Since the C implementation was derived from a functional program, all unions in seL4 are tagged, and many structs are packed bitfields. Like other kernel implementors, we do not trust GCC to compile and optimize bitfields predictably for kernel code. Instead, we wrote a small tool that takes a specification and generates C code with the necessary shifting and masking for such bitfields. The tool helps us to easily map structures to page table entries or other hardware-defined memory layouts. The generated code can

**Figure 5. C code for part of the scheduler.**

```
void setPriority(tcb_t *tptr, prio_t prio) {
    prio_t oldprio;
    if(thread_state_get_tcbQueued(tptr->tcbState)) {
        oldprio = tptr->tcbPriority;
        ksReadyQueues[oldprio] =
            tcbSchedDequeue(tptr, ksReadyQueues[oldprio]);
        if(isRunnable(tptr)) {
            ksReadyQueues[prio] =
                tcbSchedEnqueue(tptr, ksReadyQueues[prio]);
        }
    }
    else {
        thread_state_ptr_set_tcbQueued(&tptr->tcbState,
                                       false);
    }
    tptr->tcbPriority = prio;
}
```

be inlined and, after compilation on ARM, the result is more compact and faster than GCC's native bitfields. The tool not only generates the C code, it also automatically generates Isabelle/HOL specifications and proofs of correctness.

Figure 5 shows part of the implementation of the scheduling functionality described in the previous sections. It is standard C99 code with pointers, arrays and structs. The `thread_state` functions used in Figure 5 are examples of generated bitfield accessors.

#### 4.4. The proof

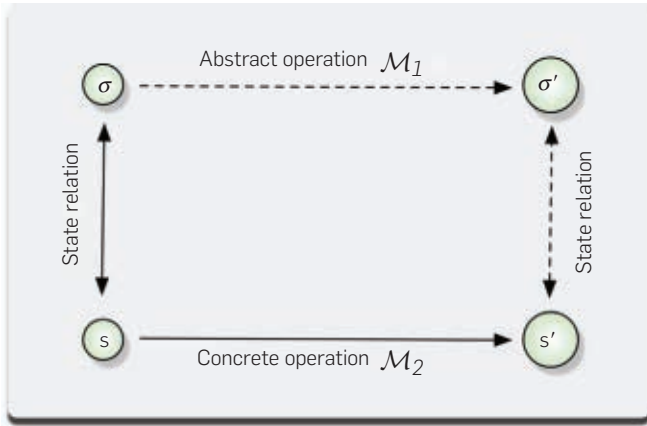
This section describes the main theorem we have shown and how its proof was constructed.

As mentioned, the main property we are interested in is functional correctness, which we prove by showing formal refinement. We have formalized this property for general state machines in Isabelle/HOL, and we instantiate each of the specifications in the previous sections into this state-machine framework.

We have also proved the well-known reduction of refinement to *forward simulation*, illustrated in Figure 6 where the solid arrows mean universal quantification and the dashed arrows existential: To show that a concrete state machine  $\mathcal{M}_2$  refines an abstract one  $\mathcal{M}_1$ , it is sufficient to show that for each transition in  $\mathcal{M}_2$  that may lead from an initial state  $s$  to a set of states  $s'$ , there exists a corresponding transition on the abstract side from an abstract state  $\sigma$  to a set  $\sigma'$  (they are sets because the machines may be nondeterministic). The transitions *correspond* if there exists a relation  $R$  between the states  $s$  and  $\sigma$  such that for each concrete state in  $s'$  there is an abstract one in  $\sigma'$  that makes  $R$  hold between them again. This has to be shown for each transition with the same overall relation  $R$ . For externally visible state, we require  $R$  to be equality. For each refinement layer in Figure 2, we have strengthened and varied this proof technique slightly, but the general idea remains the same.

We now describe the instantiation of this framework to the seL4 kernel. We have the following types of transition in

**Figure 6. Forward simulation.**



our state machines: kernel transitions, user transitions, user events, idle transitions, and idle events. *Kernel transitions* are those that are described by each of the specification layers in increasing amount of detail. *User transitions* are specified as nondeterministically changing arbitrary user-accessible parts of the state space. *User-events* model kernel entry (trap instructions, faults, interrupts). *Idle transitions* model the behavior of the idle thread. Finally, *idle events* are interrupts occurring during idle time; other interrupts that occur during kernel execution are modelled explicitly and separately in each layer of Figure 2.

The model of the machine and the model of user programs remain the same across all refinement layers; only the details of kernel behavior and kernel data structures change. The fully nondeterministic model of the user means that our proof includes all possible user behaviors, be they benign, buggy, or malicious.

Let machine  $\mathcal{M}_A$  denote the system framework instantiated with the abstract specification of Section 4.1, let machine  $\mathcal{M}_E$  represent the framework instantiated with the executable specification of Section 4.2, and let machine  $\mathcal{M}_C$  stand for the framework instantiated with the C program read into the theorem prover. Then we prove the following two, very simple-looking theorems:

**THEOREM 1.**  $\mathcal{M}_E$  refines  $\mathcal{M}_A$ .

**THEOREM 2.**  $\mathcal{M}_C$  refines  $\mathcal{M}_E$ .

Therefore, because refinement is transitive, we have

**THEOREM 3.**  $\mathcal{M}_C$  refines  $\mathcal{M}_A$ .

#### 4.5. Assumptions

Formal verification can never be absolute; it always must make fundamental assumptions. The assumptions we make are correctness of *the C compiler*, *the assembly code*, *the hardware*, and *kernel initialization*. We explain each of them in more detail below.

The *initialization code* takes up about 1.2 kloc of the kernel. The theorems in Section 4.4 only state correspondence

between entry and exit points in each specification layer for a running kernel.

Assuming correctness of the *C compiler* means that we assume GCC correctly translates the seL4 source code in our C subset according to the ISO/IEC C99 standard,<sup>4</sup> that the formal model of our C subset accurately reflects this standard and that the model makes the correct architecture-specific assumptions for the ARMv6 architecture on the Freescale i.MX31 platform.

The assumptions on *hardware and assembly* mean that we do not prove correctness of the register save/restore and the potential context switch on kernel exit. Cache consistency, cache coloring, and TLB flushing requirements are part of the assembly implemented machine interface. These machine interface functions are called from C, and we assume they do not have any effect on the memory state of the C program. This is only true if they are used correctly.

The VM subsystem of seL4 is not assumed correct, but is treated differently from other parts of the proof. For our C semantics, we assume a traditional, flat view of in-kernel memory that is kept consistent by the kernel's VM subsystem. We make this consistency argument only informally; our model does not oblige us to prove it. We do however substantiate the model and informal argument by manually stated, machine-checked properties and invariants. This means we explicitly treat in-kernel VM in the proof, but this treatment is different from the high standards in the rest of our proof where we reason from first principles and the proof forces us to be complete.

This is the set of assumptions we picked. If they are too strong for a particular purpose, many of them can be eliminated combined with other research. For instance, we have verified the executable design of the boot code in an earlier design version. For context switching, Ni et al.<sup>7</sup> report verification success, and the Verisoft project<sup>1</sup> shows how to verify assembly code and hardware interaction. Leroy verified an optimizing C compiler<sup>5</sup> for the PowerPC and ARM architectures.

An often-raised concern is the question *What if there is a mistake in the proof?* The proof is machine-checked by Isabelle/HOL. So what if there is a bug in Isabelle/HOL? The proof checking component of Isabelle is small and can be isolated from the rest of the prover. It is extremely unlikely that there is a bug in this part of the system that applies in a correctness-critical way to our proof. If there was reason for concern, a completely independent proof checker could be written in a few hundred lines of code. Provers like Isabelle/HOL can achieve a degree of proof trustworthiness that far surpasses the confidence levels we rely on in engineering or mathematics for our daily survival.

## 5. EXPERIENCE AND LESSONS LEARNED

### 5.1. Verification effort

The project was conducted in three phases. First an initial kernel with limited functionality (no interrupts, single address space, and generic linear page table) was designed and implemented in Haskell, while the verification team mostly worked on the verification framework and generic proof libraries. In a second phase, the verification team developed the abstract spec and performed the



first refinement while the development team completed the design, Haskell prototype and C implementation. The third phase consisted of extending the first refinement step to the full kernel and performing the second refinement. The overall size of the proof, including framework, libraries, and generated proofs (not shown in the table) is 200,000 lines of Isabelle script.

Table 1 gives a breakdown for the effort and size of each of the layers and proofs. About 30 person months (pm) went into the abstract specification, Haskell prototype and C implementation (over all project phases), including design, documentation, coding, and testing.

This compares well with other efforts for developing a new microkernel from scratch: The Karlsruhe team reports that, on the back of their experience from building the earlier Hazelnut kernel, the development of the Pistachio kernel costs about 6 person years (py). SLOCCount with the “embedded” profile estimates the total cost of seL4 at 4 py. Hence, there is strong evidence that the detour via Haskell did not increase the cost, but was in fact a significant net cost saver.

The cost of the proof is higher, in total about 20 py. This includes significant research and about 9 py invested in formal language frameworks, proof tools, proof automation, theorem prover extensions, and libraries. The total effort for the seL4-specific proof was 11 py.

We expect that redoing a similar verification for a new kernel, using the same overall methodology, would reduce this figure to 6 py, for a total (kernel plus proof) of 8 py. This is only twice the SLOCCount estimate for a traditionally engineered system with no assurance.

The breakdown in Table 1 of effort between the two refinement stages is illuminating: almost 3:1. This is a reflection of the low-level nature of our Haskell prototype, which captures most of the properties of the final product. This is also reflected in the proof size—the first proof step contained most of the deep semantic content. 80% of the effort in the first refinement went into establishing invariants, only 20% into the actual correspondence proof. We consider this asymmetry a significant benefit, as the executable spec is more convenient and efficient to reason about than C.

The first refinement step led to some 300 changes in the abstract spec and 200 in the executable spec. About 50% of these changes relate to bugs in the associated algorithms or design. Examples are missing checks on user-supplied input, subtle side effects in the middle of an operation breaking global invariants, or over-strong assumptions about what is true during execution. The rest of the changes were introduced for verification convenience. The

**Table 1. Code and proof statistics.**

	Haskell/C		Isabelle		Proof	
	pm	kloc	kloc	Invariants	py	klop
abst.	4	–	4.9	~75	8	110
exec.	24	5.7	13	~80	3	55
impl.	2	8.7	15	0		

ability to change and rearrange code in discussion with the design team was an important factor in the verification team’s productivity and was essential to complete the verification on time.

The second refinement stage from executable spec to C uncovered 160 bugs, 16 of which were also found during testing, early application and static analysis. The bugs discovered in this stage were mainly typos, misreading the specification, or failing to update all relevant code parts for specification changes. Even though their cause was often simple, understandable human error, their effect in many cases was sufficient to crash the kernel or create security vulnerabilities. There were no deeper, algorithmic bugs in the C level, because the C code was written according to a very precise, low-level specification.

## 5.2. The cost of change

One issue of verification is the cost of proof maintenance: how much does it cost to reverify after changes are made to the kernel? This obviously depends on the nature of the change. We are not able to precisely quantify such costs, but our iterative verification approach has provided us with some relevant experience.

The best case is a *local, low-level code change*, typically an optimization that does not affect the observable behavior. We made such changes repeatedly, and found that the effort for reverification was always low and roughly proportional to the size of the change.

*Adding new, independent features*, which do not interact in a complex way with existing features, usually has a moderate impact. For example, adding a new system call to the seL4 API that atomically batches a specific, short sequence of existing system calls took one day to design and implement. Adjusting the proof took less than 1 person week.

*Adding new, large, cross-cutting features*, such as adding a complex new data structure to the kernel supporting new API calls that interact with other parts of the kernel, is significantly more expensive. We experienced such a case when progressing from the first to the final implementation, adding interrupts, ARM page tables, and address spaces. This change costs several pms to design and implement, and resulted in 1.5–2 py to reverify. It modified about 12% of existing Haskell code, added another 37%, and reverification cost about 32% of the time previously invested in verification. The new features required only minor adjustments of existing invariants, but lead to a considerable number of new invariants for the new code. These invariants had to be preserved over the whole kernel, not just the new features.

Unsurprisingly, *fundamental changes to existing features* are bad news. We experienced one such change when we added reply capabilities for efficient RPC as an API optimization after the first refinement was completed. Even though the code size of this change was small (less than 5% of the total code base), it violated key invariants about the way capabilities were used in the system until then and the amount of *conceptual* cross-cutting was huge. It took about 1 py or 17% of the original proof effort to reverify.

There is one class of otherwise frequent code changes

that does not occur after the kernel has been verified: implementation bug fixes.

## 6. CONCLUSION

We have presented our experience in formally verifying seL4. We have shown that full, rigorous, formal verification is practically achievable for OS microkernels.

The requirements of verification force the designers to think of the simplest and cleanest way of achieving their goals. We found repeatedly that this leads to overall better design, for instance, in the decisions aimed at simplifying concurrency-related verification issues.

Our future research agenda includes verification of the assembly parts of the kernel, a multi-core version of the kernel, as well as formal verification of overall system security and safety properties, including application components. The latter now becomes much more meaningful than previously possible: application proofs can rely on the abstract, formal kernel specification that seL4 is proven to implement.

## Acknowledgments

We would like to acknowledge the contribution of the former team members on this verification project: Timothy Bourke, Jeremy Dawson, Jia Meng, Catherine Menon, and David Tsai.

NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

## References

- Alkassar, E., Schirmer, N., Starostin, A. Formal pervasive verification of a paging mechanism. *TACAS*. C.R. Ramakrishnan and J. Rehof, eds. Volume 4963 of *LNCS* (2008). Springer, 109–123.
- Dennis, J.B., Van Horn, E.C. Programming semantics for multiprogrammed computations. *CACM* 9 (1966), 143–155.
- Elkaduwe, D., Klein, G., Elphinstone, K. Verified protection model of the seL4 microkernel. *VSTTE 2008—Verified Software: Theories, Tools & Experiments*. J. Woodcock and N. Shankar eds. Volume 5295 of *LNCS* (Toronto, Canada, Oct 2008), Springer, 99–114.
- ISO/IEC. Programming languages—C. Technical Report 9899:TC2, ISO/IEC JTC1/SC22/WG14, May 2005.
- Leroy, X. Formal certification of a compiler back-end, or: Programming a compiler with a proof assistant. *33rd POPL*. J.G. Morrisett and S.L.P. Jones, eds. (New York, NY, USA, 2006), ACM, 42–54.
- Liedtke, J. Towards real microkernels. *CACM* 39, 9 (Sept 1996), 70–77.
- Ni, Z., Yu, D., Shao, Z. Using XCAP to certify realistic system code: Machine context management. *20th TPHOLS*, volume 4732 of *LNCS* (Kaiserslautern, Germany, Sept 2007), Springer, 189–206.
- Nipkow, T., Paulson, L., Wenzel, M. *Isabelle/HOL—A Proof Assistant for Higher-Order Logic*. Volume 2283 of *LNCS* (2002), Springer.
- Ormandy, T., Tinnes, J. Linux null pointer dereference due to incorrect proto\_ops initializations. <http://www.cr0.org/misc/CVE-2009-2692.txt>, 2009.
- Saltzer, J.H., Schroeder, M.D. The protection of information in computer systems. *Proc. IEEE* 63 (1975), 1278–1308.
- Shapiro, J.S., Smith, J.M., Farber, D.J. EROS: A fast capability system. *17th SOSP* (Charleston, SC, USA, Dec 1999), 170–185.

**Gerwin Klein** National ICT Australia (NICTA), University of South Wales (UNSW), Sydney.

**June Andronick** NICTA, UNSW.

**Kevin Elphinstone** NICTA, UNSW.

**Gernot Heiser** NICTA, UNSW, Open Kernel Labs.

**Dharmika Elkaduwe** NICTA, UNSW, now at University of Peradeniya, Sri Lanka.

**Kai Engelhardt** NICTA, UNSW.

**Rafal Kolanski** NICTA, UNSW.

**Harvey Tuch** NICTA, UNSW, now at VMWare.

**Simon Winwood** NICTA, UNSW.

**David Cock** NICTA.

**Philip Derrin** NICTA, now at Open Kernel Labs.

**Michael Norrish** NICTA, Australian National University, Canberra.

**Thomas Sewell** NICTA.

© 2010 ACM 0001-0782/10/0600 \$10.00

Have you  
Considered All  
the Possibilities?

Save 20%  
on these

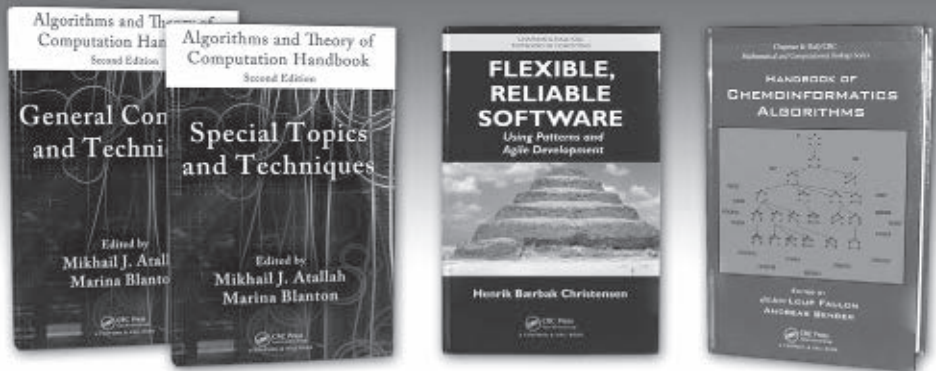
New & Noteworthy  
Resources  
from  
Chapman & Hall/CRC Press

Save 20% on your online  
order by entering promo  
code **736DM** at checkout  
Discount expires July 31, 2010

[www.crcpress.com](http://www.crcpress.com)

**CRC** CRC Press  
Taylor & Francis Group

CHAPMAN & HALL BOOKS



New Edition of a Bestseller!

**Algorithms and Theory of  
Computation Handbook,  
Second Edition—  
Two Volume Set**

Praise for the Previous Edition

“... excellent survey of the state of the art ... highly recommended for anyone interested in algorithms, data structures and the theory of computation ... an indispensable book of reference ...”  
—R. Kemp, *Zentralblatt MATH*, Vol. 926

Catalog no. C8180  
January 2010, 1938 pp.  
ISBN: 978-1-58488-818-5  
~~\$179.95 / £114.00~~  
**\$143.96 / £91.20**

**Flexible, Reliable Software  
Using Patterns and  
Agile Development**

“... brings together a careful selection of topics that are relevant, indeed crucial, for developing good quality software ... leads the reader through an experience of active learning.”  
—Michael Kölling, Bestselling

Author and originator of the BlueJ and Greenfoot environments

Catalog no. C3622  
May 2010, 527 pp.  
ISBN: 978-1-4200-9362-9  
~~\$69.95 / £44.99~~  
**\$55.96 / £35.99**

**Handbook of  
Chemoinformatics  
Algorithms**

Providing an overview of the most common chemoinformatics algorithms, this book explains how to effectively apply algorithms and graph theory for solving chemical problems.

Catalog no. C2922  
April 2010, 454 pp.  
ISBN: 978-1-4200-8292-0  
~~\$99.95 / £63.99~~  
**\$79.96 / £51.19**

# CAREERS

## **Epic** Problem Solver/Technical Consultant

Epic's Problem Solvers are responsible for our clients' happiness after the systems are installed. They create valuable relationships by listening well to customer concerns and championing clients' needs. They work with IT staff at customer sites to quickly resolve technical issues and perform necessary programming, helping to ensure that every customer gets the most out of an Epic software investment.

Candidates should have a bachelor's degree or higher (all majors considered), a minimum 3.2 cumulative GPA, and be eligible to work in the U.S. without sponsorship. No prior technical experience is necessary, but exposure to programming is a plus. Relocation to Madison, Wisconsin is required and reimbursed.

Apply URL: <http://careers.epic.com/>  
Email Address: [job@epic.com](mailto:job@epic.com)

## **Epic** Software Developer

Our small teams of software engineers participate in all aspects of the development process, from meeting customers to system design through quality assurance and delivery. Their goal is to create easy-to-use systems with optimal workflows that manage large amounts of data with sub-second response times and rock-solid stability. Our continued success in these areas is shown by Epic software systems' top-rated industry reviews. New functionality and systems are being developed daily that extend current capabilities and break new ground in the industry.

Candidates should have a bachelor's degree or higher in Computer Science, Software Engineering, or Math and a minimum 3.2 major GPA. Relocation to Madison, Wisconsin is required and reimbursed. Visa sponsorship is available.

Apply URL: <http://careers.epic.com/>  
Email Address: [job@epic.com](mailto:job@epic.com)

For more information, please visit the university website [www.niituniversity.in](http://www.niituniversity.in).

Interested applicants are invited to submit their curriculum vitae including employment history, a statement outlining research and teaching interests, list of consultancies and projects undertaken and names of at least three referees.

Applications may be sent electronically in PDF format to:  
[careers@niituniversity.in](mailto:careers@niituniversity.in)  
or the President, Dr Rajeev Shorey,  
[rajeev.shorey@niituniversity.in](mailto:rajeev.shorey@niituniversity.in)  
Phone : +91-9251083130

## **Princeton University** Computer Science Lecturer

Part- and full-time Lecturer positions. The Department of Computer Science seeks applications from outstanding teachers to assist the faculty in teaching our introductory course sequence.

The primary requirements of the job are to teach recitation sections and to participate in overall management of the introductory sequence. Other responsibilities include supervising graduate student teaching assistants and developing and maintaining online curricular material, classroom demonstrations, and laboratory exercises.

Candidates should have an exceptional record of classroom instruction and curricular innovation. An advanced degree in computer science is preferred.

For general application information and to self-identify visit: <https://jobs.princeton.edu>

Requisition Number: 1000207. You may apply online on the Department's website at: <http://www.cs.princeton.edu/jobs/lecturerposition>

We will not accept applications from the Princeton jobs site.

Princeton University is an equal opportunity employer and complies with applicable EEO and affirmative action regulations.

## **Taif University, Saudi Arabia** Assistant/Associate Professor

The College of Computers & Information Systems at TAIF University in Saudi Arabia invites applications for faculty positions for fall 2010 or spring 2011. Candidates should have Ph.D. in Computer Science/Computer Engineering/Information Systems, or a closely related discipline, and a strong commitment to excellence in teaching, research, and service.

Please email CV, letter of application, transcripts, statement of research and teaching interests, and 3 letters of reference addressing to Dean Dr. Sultan Aljahdali  
[atcisdean@tu.edu.sa](mailto:atcisdean@tu.edu.sa)

## **NIIT University, INDIA** Computer Science & Engg and Information & Communication Technology Faculty Positions in Computer Science & Engg and Information and Communication Technology

NIIT University (NU) invites applications at Assistant Professor, Associate Professor and Professor levels.

Candidates must have earned (or expect shortly) a Ph.D for Assistant Professor, Ph.D with 6 years of experience for Associate Professor, Ph.D with 10 years of experience for Professor in the above disciplines. Candidates must have a good publication record; proven ability to establish an independent research program; demonstrate flair for innovation; be open to participate in developing new interdisciplinary programs of study; have the commitment to excel both in research and teaching and establish linkage with industry. Industry experience and/or post-doctoral experience will be considered an asset.

NIIT University is located about 120 Kms from New Delhi. The University provides an intellectual environment and is committed to academic excellence. The four core principles of NIIT University, namely, Industry-linked, Research-driven, Technology-based, and Seamless define the DNA of the University.

NIIT University offers a competitive compensation at par with the best academic institutions in India. In addition, NU provides a start up research grant at the time of joining, travel support for presenting papers in International Conferences and Workshops. NU provides research incentives, such as monetary award for refereed journal publications. In line with the Industry-linked as a core principle, the university will enable faculty to consult with industry in India and abroad.



## **ADVERTISING IN CAREER OPPORTUNITIES**

**How to Submit a Classified Line Ad: Send an e-mail to [acmm mediasales@acm.org](mailto: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: Five weeks prior to the publication date of the issue (which is the first of every month). Latest deadlines: <http://www.acm.org/publications>**

**Career Opportunities Online: Classified and recruitment display ads receive a free duplicate listing on our website at: <http://campus.acm.org/careercenter>**

**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](mailto:acmm mediasales@acm.org)**



## Expansion of the Research School "Service-Oriented Systems Engineering" at Hasso-Plattner-Institute

8 Ph.D. grants available - starting October 1, 2010

Hasso-Plattner-Institute (HPI) is a privately financed institute affiliated with the University of Potsdam, Germany. The Institute's founder and benefactor Professor Hasso Plattner, who is also co-founder and chairman of the supervisory board of SAP AG, has created an opportunity for students to experience a unique education in IT systems engineering in a professional research environment with a strong practice orientation.

In 2005, HPI initiated the research school on "Service-Oriented Systems Engineering" under the scientific supervision of Professors Jürgen Döllner, Holger Giese, Robert Hirschfeld, Christoph Meinel, Felix Naumann, Hasso Plattner, Andreas Polze, Mathias Weske and Patrick Baudisch.

We are expanding our research school and are currently seeking

**8 Ph.D. students  
(monthly stipends 1400-1600 Euro)  
2 Postdocs (monthly stipend 1800 Euro)**

Positions will be available starting October 1, 2010.  
The stipends are not subject to income tax.

The main research areas in the research school at HPI are:

- Self-Adaptive Service-Oriented Systems
- Operating System Support for Service-Oriented Systems
- Architecture and Modeling of Service-Oriented Systems
- Adaptive Process Management
- Services Composition and Workflow Planning
- Security Engineering of Service-Based IT Systems
- Quantitative Analysis und Optimization of Service-Oriented Systems
- Service-Oriented Systems in 3D Computer Graphics
- Service-Oriented Geoinformatics

Prospective candidates are invited to apply with:

- Curriculum vitae and copies of degree certificates/transcripts
- A short research proposal
- Writing samples/copies of relevant scientific papers (e. g. thesis etc.)
- Letters of recommendation

Please submit your applications by July 31, 2010 to the coordinator of the research school:

**Prof. Dr. Andreas Polze, Hasso-Plattner-Institute, Universität Potsdam  
Postfach 90 04 60, 14440 Potsdam, Germany**

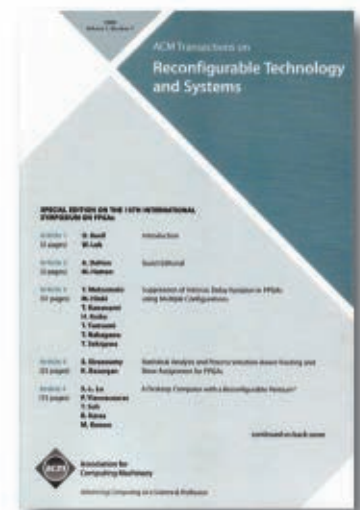
Successful candidates will be notified by September 15, 2010 and are expected to enroll into the program on October 1, 2010.

For additional information see: <http://kolleg.hpi.uni-potsdam.de>

or contact the office:  
Telephone +49-331-5509-220, Telefax +49-331-5509-229  
Email: [office-polze@hpi.uni-potsdam.de](mailto:office-polze@hpi.uni-potsdam.de)



## 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](http://www.acm.org/trets)  
[www.acm.org/subscribe](http://www.acm.org/subscribe)





Peter Winkler

DOI:10.1145/1743546.1743575

# Puzzled Solutions and Sources

Last month (May 2010, p. 120) we posted a trio of brainteasers, including one as yet unsolved, concerning variations on the Ham Sandwich Theorem.

The Intermediate Value Theorem says that if you go continuously from one real number to another, you must pass through all the real numbers in between. You can use it to prove the Ham Sandwich Theorem; here's how it can be used to solve Puzzles 1 and 2:

## 1. Hiking the Cascade Range.

**Solution.** Puzzle 1 asked us to prove that the programmers who spent Saturday climbing and Sunday descending Mt. Baker were, at some time of day, at exactly the same altitude on both days.

It's easily done. For any time  $t$ , let  $f(t)$  be the programmers' altitude on Sunday minus their altitude on Saturday;  $f(t)$  starts off positive in the morning and ends up negative at night, so at some point must be 0.

An equivalent, and perhaps more intuitive, way to see this is to imagine that the programmers have twins who were instructed to climb the mountain on Sunday exactly as the programmers climbed it the day before. Then, even if their paths up and down were different, there is some point at which the programmers and their twins must pass one another in altitude.

## 2. Inscribing a Lake in a Square.

**Solution.** Puzzle 2 asked us to show that, given any closed curve in the plane, there is a square containing the curve, all four sides of which touch the curve. The idea of the proof is both simple and elegant. Start with a vertical line drawn somewhere west of the curve. Gradually shift the line eastward until it just touches the curve. Repeat with a second line, drawn east of the curve and moving gradually west, so we now have another vertical line touching the curve on its east side. Now bring a horizontal line down from the north until it touches the curve and another from the south, thus inscribing the curve in a rectangle.

But what we want is not merely a rectangle but a square. Suppose the rectangle is taller than it is wide (as it would be in, say, Lake Champlain). Now slowly rotate the four lines together clockwise, keeping all four outside but still touching the curve. After 90 degrees of rotation, the picture is exactly the same as before, only now, the previously long vertical lines of the rectangle are the short horizontal sides.

At some point in the rotation process, the original vertical lines and horizontal lines were all the same length—and, at exactly that point, the curve was inscribed in a square.

## 3. Curves Containing the Corners of a Square.

**Solution.** The third puzzle was (as usual) unsolved, frustrating geometers for more than a century. For a discussion see <http://www.ics.uci.edu/~eppstein/junkyard/jordan-square.html>, including reference to an article by mathematician Walter Stromquist ("Inscribed Squares and Square-like Quadrilaterals in Closed Curves," *Mathematika* 36, 2 (1989), 187–197) in which he proved the conjecture for smooth curves. See also Stan Wagon's and Victor Klee's book *Old and New Unsolved Problems in Plane Geometry and Number Theory* (Mathematical Association of America, 1991).

All readers are encouraged to submit prospective puzzles for future columns to [puzzled@cacm.acm.org](mailto:puzzled@cacm.acm.org).

**Peter Winkler** ([puzzled@cacm.acm.org](mailto:puzzled@cacm.acm.org)) is Professor of Mathematics and of Computer Science and Albert Bradley Third Century Professor in the Sciences at Dartmouth College, Hanover, NH.

[CONTINUED FROM P. 120] all potential competitors. Crude versions of the same pattern are seen among chimps, wolves, and many other species. The logic of Darwin and Malthus may be pervasive on other worlds, too. Could this help explain the daunting sky-silence?

One important exception was by far the most successful Earth-based civilization—the Scientific Enlightenment—which broke from the ancient feudal pattern, fostering instead what Robert Wright, author of *Nonzero: The Logic of Human Destiny*, called the “positive sum game,” encouraging individualism and copious self-criticism. (It’s a theme typified by self-reproachful messages like James Cameron’s movie *Avatar*.) What better way to detect, reveal, and resolve myriad potential pitfalls than by unleashing millions of diverse, highly educated, technologically empowered citizens to swarm (like T-cells) on every apparent failure mode, real or imagined?

This noisy process was supercharged in the late 1980s when the U.S. government did something that still seems historically anomalous: releasing the Internet it had invented from near-total control, simply handing it over to the world. Ponder, in the light of the past 4,000 years of recorded history, the likelihood of such a decision. Was there ever anything comparable, under the most beneficent kings?

I want to defend this rambunctious culture of freedom for all the usual reasons (such as “freedom is good!”), but I wonder. Such an experiment was rare here on Earth and seems unlikely to have been tried very often, out there across the cosmos. In fact (and here’s the point of this digression), our latest, tech-amplified version of the Enlightenment could become a fiercely effective problem-solving system helping us become the exception... a sapient race that survives. That is, if it is allowed to. And if it ever matures.

But getting the most from our potential also requires better tools. In his book *Smart Mobs* (2002), social scholar Howard Rheingold envisioned a future when the savvy, liberated populace forms resilient, ad hoc, problem-solving networks that pounce on errors and dangers, adapting much quicker than stuffy, traditional, hierarchical institutions. Recall that citizen-power was the

**Success may depend on new skills and tools that empower our “adolescent” traits, the drive that makes us hunger for adventure, surprise, even fun.**

only thing that worked well on 9/11. We’ll surely need such agility and initiative in times to come.

Still, no one has yet disproved the hoary adage that “The intelligence of a crowd is that of its dumbest member, divided by the number in the crowd.” Could any blog, social-networking site, or twit-mesh be described as a “problem-solving discourse”? Not unless you have very low standards of “discourse.” Today’s communications platforms seem obstinately, even proudly, primitive, encouraging dumb-down groupthink that Jaron Lanier called “digital Maoism” in his Future Tense essay “Confusions of the Hive Mind” (Sept. 2009), not the vigorously new-citizenship I forecast in my 1989 eco-thriller *Earth*.

Connectivity scholar and Google Vice President Marissa Mayer says the Internet is in its “adolescence.” Indeed, many of the traits tech-zealot Clay Shirky (<http://www.shirky.com/>) adores and that Web critic Nicholas Carr (<http://www.rough.type.com/>) abhors are qualities we associate with our own teenage years. Take the rampant flightiness of scattered attention spans, simplistic online tribalism, and tsunamis of irate opinion; picture 10 million electronic Nuremberg rallies. These punk attributes blend and contrast with positive adolescent qualities like unprecedented vividness, creativity, quickness, alert compassion, and spontaneity. No generation ever read or wrote so much... albeit, never was such a high fraction of the writing such drivel.

There’s nothing wrong with self-

expression. Not everyone is required to engage in erudite discourse. But must the medium conspire to make discourse next to impossible, leaving each decade’s version of “conversation” more terse and lobotomized? Must the interface assume that superficiality is the chief desideratum and self-fulfilling expectation?

If this is an “adolescent phase,” we may yet see what Wikipedia co-founder Larry Sanger calls “sophrosyne,” or polemical shouting transforming into fair disputation and negotiation, a trick many teens eventually learn.

Imagine today’s Internet augmented by a shopping list of now-missing tools to enhance attention allocation, empowering users to do more in parallel while rediscovering the art of concentration. Today’s fetish for “gisting,” or grabbing the summarized essence of any fact or opinion, might yet be more useful and accurate, when coupled with utilities for source-reputation weighting, paraphrasing, correlation, data analysis, and what Howard Rheingold called general “crap-detection.” Collaborationware might yet evolve from its present stodginess, helping ad hoc teams self-organize, divide tasks, delegate expertise, and achieve quick wonders. Such tools could start by bringing online some of the amazing mental methods we take for granted in the real world (such as the way we sift for meaning from multiple conversations at once, as at a cocktail party). Many have never been implemented online, in any way.

Destiny, not only on Earth but across the Galaxy, may depend on how we choose to cross this danger zone. Success could arise less out of stodgy prescriptions than from those “adolescent” traits that make us hunger for adventure, surprise, even fun. Only... perhaps empowered by new skills that help us function as *thoughtful* adolescents, more like precocious 19-year-olds than scatterbrained 13-year-olds. Perhaps even like people with grownup goals... and the patience to achieve them. ■

**David Brin** (<http://www.davidbrin.com>) is a scientist, technology speaker, and author whose stories and novels have won Hugo and Nebula awards. He is also the author of the nonfiction book *The Transparent Society: Will Technology Make Us Choose Between Freedom and Privacy?* (Perseus Books Group, 1989).

© 2010 ACM 0001-0782/10/0600 \$10.00



Future Tense, one of the revolving features on this page, presents stories and essays from the intersection of computational science and technological speculation, their boundaries limited only by our ability to imagine what will and could be.

DOI:10.1145/1743546.1743576

David Brin

# Future Tense

## How the Net Ensures Our Cosmic Survival

*Give adolescent wonder an evolutionary jolt.*

THE INTERNET HAS changed the way I think, though, ironically, less than I expected. As both a freelance scientist and a science fiction author, I already telecommuted back in 1980, kept flexible hours, digitally collaborated with colleagues around the world, conducted digital literature searches, and was an early adopter of text editing. All these trends have since accelerated. Yet, compared to my colleagues' utopian visions for 2010, today's Net and Web remain, a bit, well, stodgy.

Oh, I'm grateful to live in such times. For one thing, without the Internet, civilization would likely have fallen into the Specialization Trap that tech students pondered, pessimistically, in the 1960s. At the time, it seemed inevitable—as the weight of accumulated knowledge piled higher and higher—that researchers would have to learn more and more about less and less, in narrowing subfields, staggering forward ever more slowly under the growing burden of human progress. Specialty boundaries would grow more rigid and uncrossable. This unpleasant fate seemed unavoidable, back when “information” had a heavy, almost solid quality...

...till the Internet Era transformed knowledge into something more like a gas—or sparkling plasma—infininitely malleable, duplicable, accessible, mobile. At which point the old worries about death-by-overspecialization vanished so thoroughly that few recall how gloomy the prospect seemed, only a few decades ago.

Today, some fear the opposite failure mode, veering from narrow-mind-

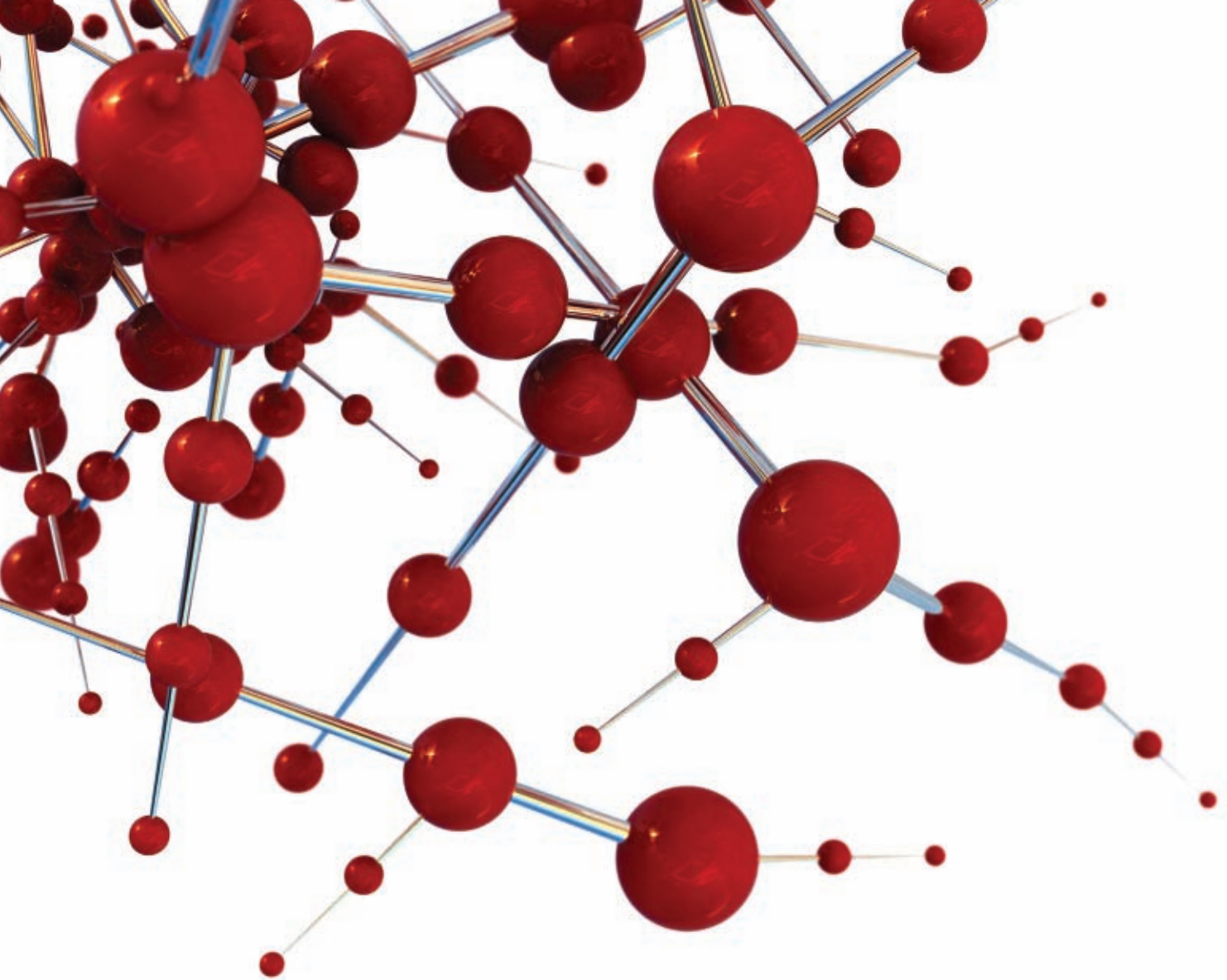


ed overspecialization to scatterbrained shallowmindedness. Flitting about info-space, we snatch one-sentence summaries of anything that's known by the vast, collective mind. Whatever the topic, each of us is able to preen with presumptuous “expertise.” This trend colors even modern politics; a core tenet of the Culture War holds that specialists are no more qualified than opinionated amateurs to judge truth.

Worrisome trends have always seemed to threaten civilization. From Plato, Gibbon, and Spengler to Toynbee, Kennedy, and Diamond, many have diagnosed why cultures succeed or fail. Theories vary, but the implications go far beyond the fate of mere humanity. In his Future Tense essay “Radical Evolution” (Mar. 2009), Joel Garreau

tied technology's march to the question of why we've seen no signs of intelligent life beyond planet Earth, not even radio blips on a SETI screen. Does this Great Silence suggest every sapient race out there ultimately repeats the same technology-driven mistakes, driving their own civilizations to ruin?

We know next to nothing about aliens but can impute something about them from the self-perpetuating instinctive drives that propel both human societies and nearly all animal species on Earth, spurred by the “zero sum,” or I-win-by-making-you-lose, logic of reproductive success. Hence, 99% of human cultures that ever achieved farming and metals also wound up ruled by feudal oligarchies that squelched [CONTINUED ON P. 119]



**CONNECT WITH OUR  
COMMUNITY OF EXPERTS.**

**[www.reviews.com](http://www.reviews.com)**



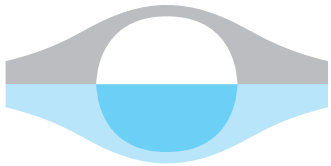
Association for  
Computing Machinery

**Reviews.com**

They'll help you find the best new books  
and articles in computing.

**Computing Reviews is a collaboration between the ACM and Reviews.com.**





**CSCW2011**  
BUILDING BRIDGES

# The Twenty-Third ACM Conference on Computer Supported Cooperative Work

March 19–23, 2011 · Hangzhou, China

CSCW is an international and interdisciplinary conference that focuses on how technology intersects with social practices. Join us in **Building Bridges** by connecting with social and technical researchers at the conference and interacting with research communities from around the world.

#### Conference Co-chairs

Pamela Hinds, *Stanford University*

John C. Tang, *Microsoft Research*

Jian Wang, *Alibaba Group*

#### Submit by August 6, 2010

Papers & Notes  
Showcase Papers  
Workshops  
Tutorials

#### Submit by October 1

Doctoral Colloquium  
Panels

#### Submit by November 19

Interactive Papers  
Demonstrations  
CSCW Horizon  
Videos

[www.flickr.com/photos/pedronet/2790877585/](http://www.flickr.com/photos/pedronet/2790877585/)

Sponsored by



**SIGCHI**  
special interest group computer human interaction

Find CSCW 2011 online!

[www.cscw2011.org](http://www.cscw2011.org)

[www.facebook.com/cscw2011](http://www.facebook.com/cscw2011)

[www.twitter.com/cscw2011](http://www.twitter.com/cscw2011)