

COMMUNICATIONS

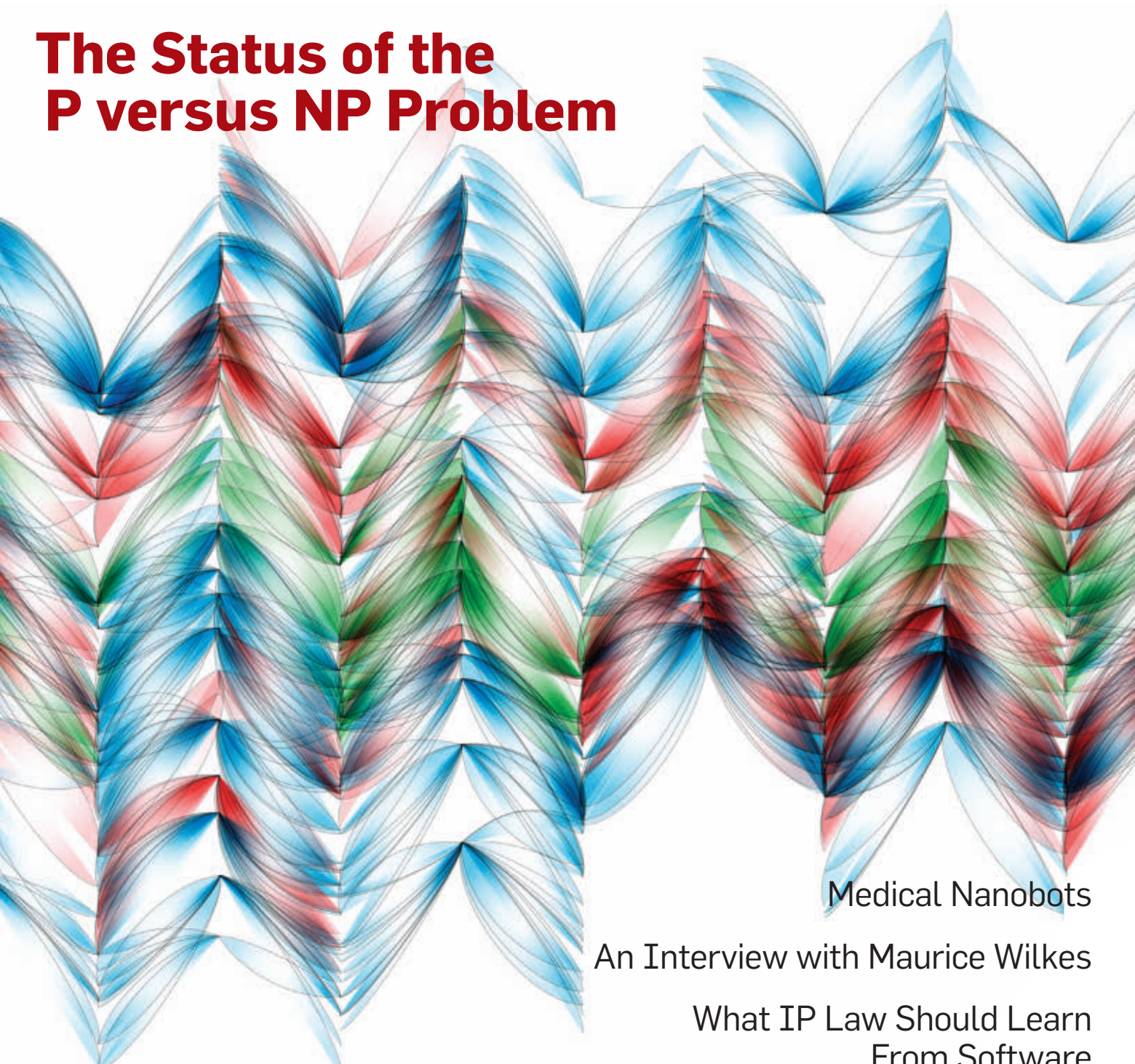
CACM.ACM.ORG

OF THE

ACM

09/2009 VOL.52 NO.09

The Status of the P versus NP Problem



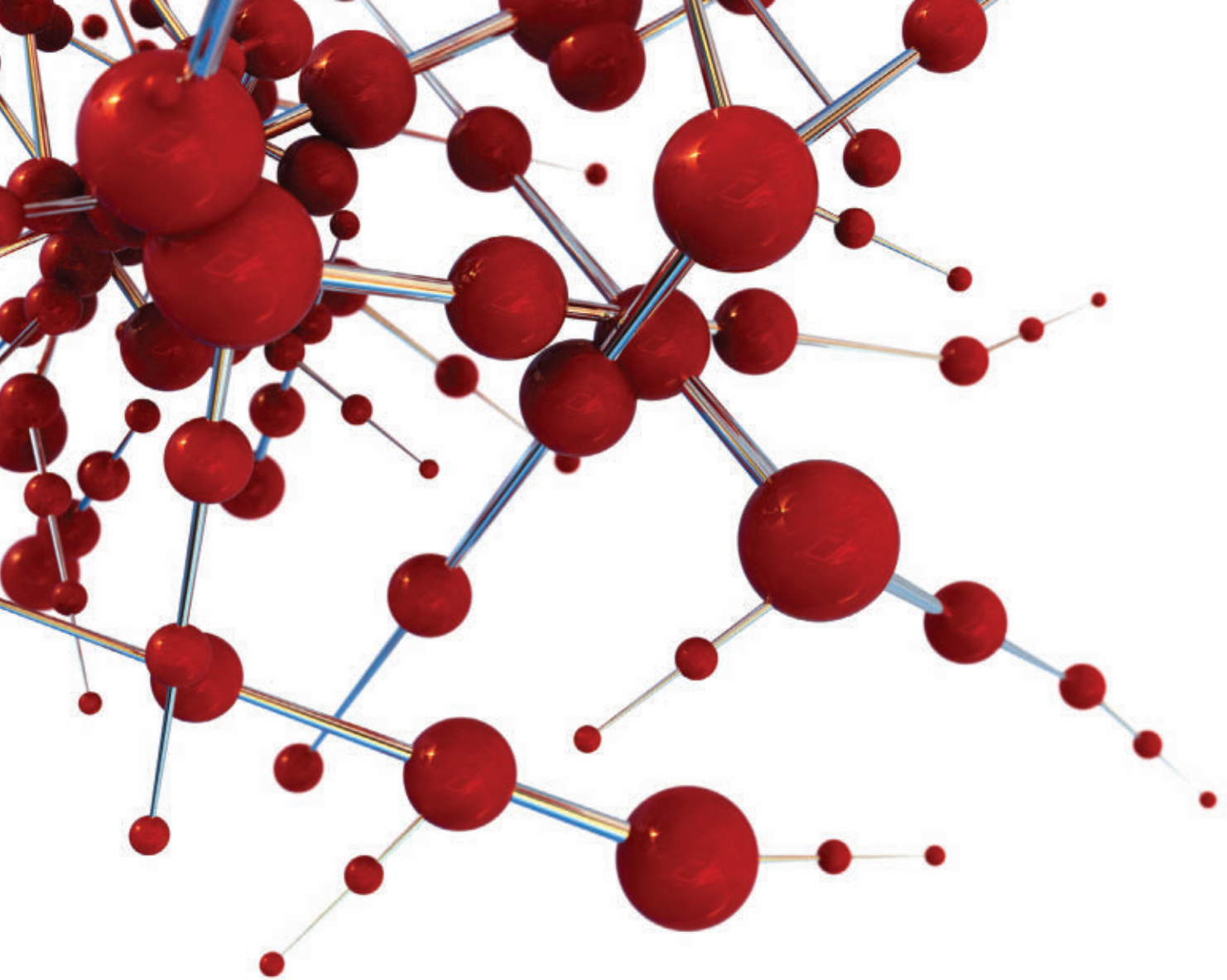
Medical Nanobots

An Interview with Maurice Wilkes

What IP Law Should Learn
From Software

Spamalytics





**CONNECT WITH OUR
COMMUNITY OF EXPERTS.**

www.reviews.com



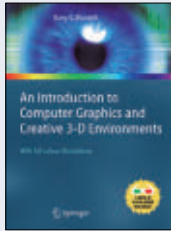
Association for
Computing Machinery

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.

Computer Science Textbooks



An Introduction to Computer Graphics and Creative 3-D Environments

B. G. Blundell, Auckland University of Technology, Auckland, New Zealand

This book introduces the fundamentals of 2-D and 3-D computer graphics. Additionally, a range of emerging, creative 3-D display technologies are described, including stereoscopic systems, immersive virtual reality, volumetric, varifocal, and others. Designed to be a core teaching text at the undergraduate level, accessible to students with wide-ranging backgrounds, only an elementary grounding in mathematics is assumed as key maths is provided.

2008. XX, 480 p. 248 illus., 227 in color. Hardcover
ISBN 978-1-84800-041-4 ► **\$69.95**



The Semantic Web

Semantics for Data and Services on the Web

V. Kashyap, Partners HealthCare System, Wellesley, MA, USA; **C.**

Bussler, Merced Systems Inc., Redwood Shores, CA, USA; **M. Moran**, Nortel, Galway, Ireland

The Semantic Web is a vision – the idea of having data on the Web defined and linked in such a way that it can be used by machines not just for display purposes but for automation, integration and reuse of data across various applications. Technically, however, there is a widespread misconception that the Semantic Web is primarily a rehash of existing AI and database work focused on encoding knowledge representation formalisms in markup languages such as RDF(S), DAML+OIL or OWL.

2008. XVI, 414 p. 61 illus. (Data-Centric Systems and Applications) Hardcover
ISBN 978-3-540-76451-9 ► **\$79.95**



Foundations of 3D Graphics Programming

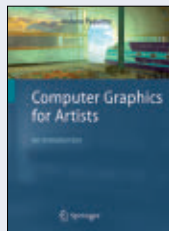
Using JOGL and Java3D

J. X. Chen, George Mason University, Fairfax, VA, USA; **C. Chen**,

Southwest Jiaotong University, Sichuan, China

This second edition contains 3 new chapters, a new appendix and is updated and enhanced throughout Supreme reference on JOGL programming, with extensive and complete examples Also covers Java3D, with detailed example programs Serves as an ideal shortcut to 3D graphics theory Written by a recognized leader in 3D graphics, virtual experiences and statistical data visualization, based on years of teaching and research experience

2nd ed. 2008. XVI, 400 p. 141 illus., 40 in color. Hardcover
ISBN 978-1-84800-283-8 ► **\$79.95**

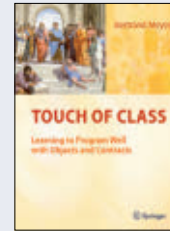


Computer Graphics for Artists: An Introduction

A. Paquette, School of Game Architecture and Design, Breda, The Netherlands

Computer Graphics for Artists: an introduction is an application-independent, reader-friendly primer for anyone with a serious desire to understand 3D Computer Graphics. Opening with the first and most basic elements of computer graphics, the book rapidly advances into progressively more complex concepts. Each of the elements, however simple, are important to understand because each is an essential link in a chain that allows an artist to master any computer graphics application.

2008. XVIII, 270 p. 453 illus., 222 in color. Softcover
ISBN 978-1-84800-140-4 ► **\$59.95**



Touch of Class

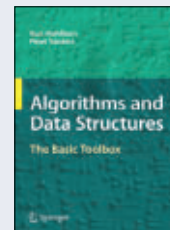
Learning to Program Well with Objects and Contracts

B. Meyer, ETH Zurich, Switzerland

From object technology pioneer and ETH Zurich

professor Bertrand Meyer, winner of the Jolt award and the ACM Software System Award, a revolutionary textbook that makes learning programming fun and rewarding. Meyer builds his presentation on a rich object-oriented software system supporting graphics and multimedia, which students can use to produce impressive applications from day one, then understand inside out as they learn new programming techniques.

2009. Approx. 700 p. Hardcover
ISBN 978-3-540-92144-8 ► **\$79.95**



Algorithms and Data Structures

The Basic Toolbox

K. Mehlhorn, Max-Planck-Institut für Informatik, Saarbrücken, Germany; **P. Sanders**, University of Karlsruhe, Germany

This book is a concise introduction addressed to students and professionals familiar with programming and basic mathematical language. Individual chapters cover arrays and linked lists, hash tables and associative arrays, sorting and selection, priority queues, sorted sequences, graph representation, graph traversal, shortest paths, minimum spanning trees, and optimization.

2008. XII, 300 p. 112 illus. Hardcover
ISBN 978-3-540-77977-3 ► **\$44.95**

Departments

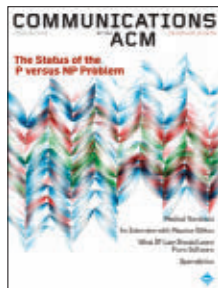
- 5 **Editor's Letter**
The Financial Meltdown and Computing
By Moshe Y. Vardi
-
- 8 **Letters to the Editor**
Computer Science Does Matter
-
- 10 **In the Virtual Extension**
-
- 12 **blog@CACM**
Saying Good-bye to DBMSs, Designing Effective Interfaces
 Michael Stonebraker discusses the problems with relational database management systems and possible solutions, and Jason Hong writes about interfaces and usable privacy and security.
-
- 14 **CACM Online**
What You Read on Your Summer Vacation
By David Roman
-
- 37 **Calendar**
-
- 108 **Careers**

Last Byte

- 110 **Puzzled**
Solutions and Sources
By Peter Winkler
-
- 112 **Future Tense**
Confusions of the Hive Mind
By Jaron Lanier

News

- 15 **Entering a Parallel Universe**
 The multicore processors that help extend Moore's Law may run afoul of Amdahl's Law.
By Gregory Goth
-
- 18 **Medical Nanobots**
 Researchers working in medical nanorobotics are creating technologies that could lead to novel health-care applications, such as new ways of accessing areas of the human body that would otherwise be unreachable without invasive surgery.
By Kirk L. Kroeker
-
- 20 **Facing an Age-Old Problem**
 Researchers are addressing the computing challenges of older individuals, whose needs are different—and too often disregarded.
By Samuel Greengard
-
- 23 **Computer Science Meets Environmental Science**
 Scientists share knowledge and seek collaborators at computational sustainability conference.
By Karen A. Frenkel



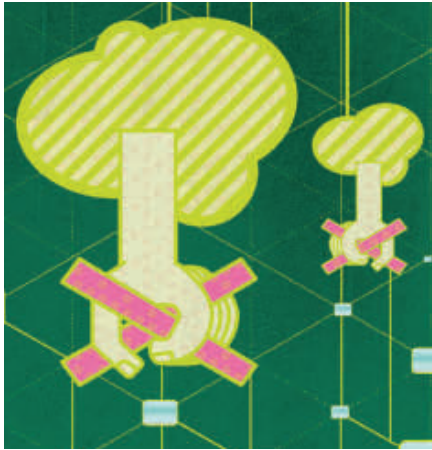
creating images, animation, and interaction.

About the Cover:
 C.E.B. Reas lives and works in Los Angeles. He is a professor in the Department of Design Media Arts at the University of California, Los Angeles. His work has been exhibited internationally. In 2001 with Ben Fry, Reas initiated Processing, an open source programming language and environment for

Viewpoints

- 24 **Law and Technology**
Keeping Track of Telecommunications Surveillance
 The creation of a statistical index of U.S. telecommunications surveillance activities and their results will benefit both civil liberties and law enforcement.
By Paul M. Schwartz
-
- 27 **The Profession of IT**
Computing: The Fourth Great Domain of Science
 Computing is as fundamental as the physical, life, and social sciences.
By Peter J. Denning and Paul S. Rosenbloom
-
- 30 **Emerging Markets**
How ICT Advances Might Help Developing Nations
 Some predictions for technology developments, deployments, and the associated societal implications.
By Mark Cleverley
-
- 33 **IT Policy**
The Long Road to Computer Science Education Reform
 Viewing the factors impeding improvements to CS education from kindergarten through grade 12 from a policy perspective.
By Cameron Wilson and Peter Harsha
-
- 36 **Viewpoint**
Face the Inevitable, Embrace Parallelism
 Hardware, software, and applications must all evolve in anticipation of the proliferation of parallelism.
By Anwar Ghuloum
-
- 39 **Interview**
An Interview with Maurice Wilkes
 Maurice Wilkes, the designer and builder of the EDSAC—the first computer with an internally stored program—reflects on his career.
By David P. Anderson

Practice



- 44 **Reveling in Constraints**
The Google Web Toolkit is an end-run around Web development obstacles.
By Bruce Johnson
-
- 49 **Monitoring and Control of Large Systems with MonALISA**
MonALISA developers describe how it works, the key design principles behind it, and the biggest technical challenges in building it.
By Iosif Legrand, Ramiro Voicu, Catalin Cirstoiu, Costin Grigoras, Latchezar Betev, and Alexandru Costan
-
- 56 **Making Sense of Revision-Control Systems**
All revision-control systems come with complicated sets of trade-offs. How do you find the best match between tool and team?
By Bryan O'Sullivan



Article development led by acmqueue.queue.acm.org

Contributed Articles

- 64 **Sound Index: Charts For the People, By the People**
Mining the wisdom of the online crowds generates music business intelligence, identifying what's hot and what's not.
By Varun Bhagwan, Tyrone Grandison, and Daniel Gruhl
-
- 71 **What Intellectual Property Law Should Learn from Software**
Software's close encounters with the law provide some lessons for our future.
By James Boyle

Review Article

- 78 **The Status of the P versus NP Problem**
It's one of the fundamental mathematical problems of our time, and its importance grows with the rise of powerful computers.
By Lance Fortnow

Research Highlights

- 88 **Technical Perspective**
Abstraction for Parallelism
By Katherine Yelick
-
- 89 **Optimistic Parallelism Requires Abstractions**
By Milind Kulkarni, Keshav Pingali, Bruce Walter, Ganesh Ramanarayanan, Kavita Bala, and L. Paul Chew
-
- 98 **Technical Perspective**
They Do Click, Don't They?
By Marc Dacier
-
- 99 **Spamalytics: An Empirical Analysis of Spam Marketing Conversion**
By Chris Kanich, Christian Kreibich, Kirill Levchenko, Brandon Enright, Geoffrey M. Voelker, Vern Paxson, and Stefan Savage

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.

Ballot Box Communication in Online Communities

Mu Xia, Yun Huang, Wenjing Duan, and Andrew B. Whinston

Examining User Involvement in Continuous Software Development

Achita (Mi) Muthitacharoen and Khawaja A. Saeed

Constructive Function-based Modeling in Multilevel Education

Alexander Pasko and Valery Adzhiev

One Size Does Not Fit All: Legal Protection for Non-Copyrightable Data

Hongwei Zhu and Stuart E. Madnick

The State of Corporate Web Site Accessibility

Eleanor T. Loiacono, Nicholas C. Romano, Jr., and Scott McCoy

Reducing Employee Computer Crime through Situational Crime Prevention

Robert Willison and Mikko Siponen

Modified Agile Practices for Outsourced Software Projects

Dinesh Batra

Technical Opinion**Falling into the Net: Main Street America Playing Games and Making Friends Online**

James Katz and Ronald E. Rice



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 Cappo

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
 Gul Agha; Michel Beaudouin-Lafon;
 Jack Davidson; Nikil Dutt; Carol Hutchins;
 Ee-Peng Lim; M. Tamer Ozsu; Vincent Shen; Mary Lou Soffa; Ricardo Baeza-Yates

ACM U.S. Public Policy Office

Cameron Wilson, Director
 1100 Seventeenth St., NW, Suite 50
 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

GROUP PUBLISHER
 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 acmmediasales@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; Richard Ladner;
 Susan Landau; Carlos Jose Pereira de Lucena;
 Helen Nissenbaum; Beng Chin Ooi;
 Loren Terveen

PRACTICE

Chair
 Stephen Bourne
Board Members
 Eric Allman; Charles Beeler;
 David J. Brown; Bryan Cantrill;
 Terry Coatta; Mark Compton;
 Benjamin Fried; Pat Hanrahan;
 Marshall Kirk McKusick;
 George Neville-Neil
 The Practice section of the CACM
 Editorial Board also serves as
 the Editorial Board of [queue](http://queue.acm.org).

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;
 Linda Petzold; Michael Reiter;
 Mendel Rosenblum; Ronitt Rubinfeld;
 David Salesin; Lawrence K. Saul;
 Guy Steele, Jr.; Gerhard Weikum;
 Alexander L. Wolf

WEB
Co-chairs
 Marti Hearst and James Landay
Board Members
 Jason I. Hong; Jeff Johnson;
 Greg Linden; Wendy E. MacKay



ACM Copyright Notice

Copyright © 2009 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.



Moshe Y. Vardi

DOI:10.1145/1562164.1562165

The Financial Meltdown and Computing

For many of us, the past year has been one of the most unsettling in our lifetime. In the late 1980s and early 1990s, we watched communism collapse of its own dead weight. In late 2008, we saw capitalism nearly crumble.

Lehman Brothers, a major U.S. investment bank, declared bankruptcy last September, sending the world's financial system into a tailspin. Only a massive intervention by central banks saved the system from collapse.

Many reasons have been offered for the near-collapse of the global economy: Alan Greenspan kept interest rates too low too long, greedy lenders pushed subprime loans on unqualified borrowers, and so on. A common thread to these explanations is that our financial system harbored a systemic risk that evaded attention until it was too late. From the perspective of computing professionals, the crisis was caused by "them," and we are its hapless victims. I'd like to offer here another explanation. I think information technology played a major role in the crisis.

The latest financial crisis is the third in the last 25 years. In Oct. 1987, stock markets around the world crashed. That crisis was blamed on program trading, which is trading driven by computer programs, implementing arbitrage and portfolio-insurance strategies. Ten years later, the Asian Financial Crisis hit mostly Asian markets, but led to the bailout of Long-Term Capital Management, a large U.S. hedge fund, whose arbitrage strategies threatened the stability of the U.S. financial market. No other period has witnessed financial crises with such frequency. A common thread to these disasters is that our financial system has reached a level of

complexity that makes it opaque even to the "high priests" of finance.

You may recall the "computer-productivity paradox" of the late 1980s and early 1990s, referring to the gap between the level of investment in information technology and the slow growth of productivity. I always thought that discussions of this "paradox" often miss an important point. To assess the value of information technology to the economy, one must contemplate how the economy would have fared without it. The obvious answer is that today's complex economic world would simply be infeasible without information technology.

In his 1986 book, *The Control Revolution: Technological and Economic Origins of the Information Society*, James Beniger showed how the introduction of railroads and the telegraph in the 19th century enabled the growing complexity of the economy. JoAnne Yates described the intimate connection between information technology and economic complexity in her 1989 book, *Control through Communication: The Rise of System in American Management*. Modern technology has enhanced this trend to the point that Bruce Lindsay, a well-known IBM database researcher, recently quipped that "relational databases form the bedrock of Western civilization." Indeed, if a massive electromagnetic pulse wiped out our computing infrastructure, our society would face a catastrophic collapse.

Information technology has enabled the development of a global financial system of incredible sophistication. At the same time, it has enabled the development of a global financial system of such complexity that our ability to comprehend it and assess risk, both localized and systemic, is severely limited. Financial-oversight reform is now a topic of great discussion. The focus of these talks is primarily over the structure and authority of regulatory agencies. Little attention has been given to what I consider a key issue—the opaqueness of our financial system—which is driven by its fantastic complexity. The problem is not a lack of models. To the contrary, the proliferation of models may have created an illusion of understanding and control, as is argued in a recent report titled "The Financial Crisis and the Systemic Failure of Academic Economics."

The question for computing as a discipline is whether we have something to contribute to this discussion. Our technology has enabled the development of this highly complex system, but can it help penetrate this complexity? Can we build computational models for the global financial system that would help us understand rather than obscure its behavior? Most importantly, I believe, we must understand that technology has societal consequences; it played a key role in creating the mess we are in. It is not only "them," it is also "us."

Moshe Y. Vardi, EDITOR-IN-CHIEF

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 world-wide, 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



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:

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

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

Fax
+1-212-944-1318

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

Special rates for residents of developing countries:

<http://www.acm.org/membership/L2-3/>

Special rates for members of sister societies:

<http://www.acm.org/membership/dues.html>

Please print clearly

Name _____

Address _____

City _____ State/Province _____ Postal code/Zip _____

Country _____ E-mail address _____

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

Purposes of ACM

ACM is dedicated to:

- 1) advancing the art, science, engineering, and application of information technology
- 2) fostering the open interchange of information to serve both professionals and the public
- 3) promoting the highest professional and ethics standards

I agree with the Purposes of ACM:

Signature _____

ACM Code of Ethics:

<http://www.acm.org/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

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
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 _____

Computer Science *Does* Matter

IT WAS DISAPPOINTING that two competent computer scientists—Matthias Felleisen and Shiram Krishnamurthi—took such a narrow view in their Viewpoint “Why Computer Science Doesn’t Matter” (July 2009). For them programming is apparently the essence of computer science, at its best when coupled with mathematics practice; therefore the science and engineering don’t matter in the contest for young minds in high school.

They wrote, but did not justify, that “programming is our field’s single most valuable skill” (what about the ability to abstract?); that graphics is “frosting” (really?); that the “three Rs” are the driving force in the K–12 curriculum (a view no longer shared by leading educators); and that leading CS educators want to emphasize engineering and science while marginalizing programming (say again?). This odd amalgam of unfounded assumptions led them to the untenable conclusion that hiding computing in a mathematics curriculum strengthens CS in high schools.

Their extended example of “imaginative programming” illustrates a genuinely creative way to make high school mathematics more engaging. It also seemed to be an argument for more CS, not less; CS and computational thinking can provide students valuable concepts and frameworks for understanding complex subjects while making courses more interesting to them.

I agree with them that the ETS decision to abolish AP tests in CS was deplorable, and we should all be working to reverse it.

Education leaders have long known that the best way to get K–12 schools to teach a particular subject is for universities to require that their students have prior education in that subject. This strategy has been particularly effective in California, which has a dominant, respected university system. If we want entering freshman to have more rigorous preparation in computing, we

need to work on our respective universities to require it. In some cases, it may be as simple as instituting the requirement in one’s own department.

Felleisen’s and Krishnamurthi’s exhortation to hide behind mathematics makes no sense. We should instead be proactively making the case not only for CS *qua* CS but for CS as a powerful conceptual tool in a variety of endeavors.

They were right to call attention to the early stages of the university curriculum. Ignoring them, CS will never attract the students it wants. Here, because the definition of CS comes to the fore, consider two principles: Insisting on a narrow definition of a field is not a sound idea in a period of growth and discovery, as it limits innovation. And we know from experience and research that the best way to motivate students to learn the fundamentals is to show them how they are used. A broad definition and relevant examples help all students, especially those who have decided to not major in computing.

Peter A. Freeman, Atlanta, GA

Matthias Felleisen and Shiram Krishnamurthi (July 2009) referred to my and Andrew McGettrick’s “The Profession of IT” column “Recentering Computer Science” (Nov. 2005), claiming we sought to marginalize “our field’s most valuable skill (programming).” That is not what we sought or accomplished. We wrote because we were deeply concerned about an external view of the field that marginalized us because it deeply misunderstands what we mean by programming. We speculated that the public perception that CS = programming, coupled with the narrow public view of the definition of “programming,” cast the entire field in a poor light. We wrote that programming is an essential core practice that won’t disappear. We suggested that recentering our own thinking (making it less focused on programming while including more engineering and science) would help

build a much stronger curriculum and public image.

Peter J. Denning (past president of ACM), Monterey, CA

Authors Respond:

Freeman correctly recognizes that we called for additional CS education, not less. He can certainly try to convince schools to introduce computing but will likely meet the response that it’s already provided in the courses that teach Word and Excel. This is indeed what we’ve learned from our 14 years in the trenches of outreach.

Our vision offers students a strong foundation for abstraction, engineering, and science—with compelling content from the bottom up, not by fiat imposed from the top down.

Matthia Felleisen, Boston

Shriram Krishnamurthi, Providence, RI

ACM Content Wants to Be Free

Addressing the question of why ACM doesn’t adopt the open-access model for its publications in his Editor’s Letter “Open, Closed, or Clopen Access?” (July 2009), Moshe Y. Vardi wrote that “‘free’ is not a sound business model.”

Though he was rebutting the conventional wisdom that “information wants to be free,” here the word “free” meant freedom, not price. Freedom is not a sound business model. It is not a business model at all but rather a mode of social interaction that human beings value and aspire to achieve.

Moreover, ACM is not a conventional business enterprise, describing itself, right on the *Communications* masthead, as “the world’s largest educational and scientific computing society.” As such, its mission is not to generate profits by implementing business models, sound or unsound, but to promote the open exchange of ideas. This means ACM publications should be, as defined by the Budapest Open Access Initiative (<http://www.soros.org/openaccess/read.shtml>), available to the public, so everyone is able to “read, download, copy, distribute,

print, search, or link to the full texts of these articles, or use them for any other lawful purpose, without financial, legal, or technical barriers other than those inseparable from gaining access to the Internet itself." Vardi, to his credit, is a signer of the Initiative. I urge him to reconsider its implications for ACM journals.

His more general point was that the status quo should be good enough, since the price of ACM's publications is, in his words, "very reasonable." He invited readers to consult their librarians for confirmation. My librarian, Kevin Engel of the Kistler Science Library at Grinnell College, says that the prices charged for ACM publications are roughly comparable to those of other professional organizations, perhaps not quite as outrageous as those of, say, the American Chemical Society, though somewhat more outrageous than the American Psychological Association.

However, Engel also says that ACM stands out as the only professional organization that views the print version of its publications as its main profit center and hence refuses to offer an online-only subscription bundle, putting many researchers and students at a disadvantage.

As a signer of the Budapest Initiative, I strongly prefer open access to ACM journals for everyone. Even if ACM is unwilling to take such a step, we could move incrementally in this general direction by unbundling its subscriptions and making the online editions of its publications separately available for purchase.

John David Stone, Grinnell, IA

I'd like to thank Moshe Y. Vardi for his thoughtful analysis of open access (July 2009) and offer a few additional points. (My employer, O'Reilly Media, provides technical information in the form of books and other media in both open and closed forms; this comment does not necessarily represent the views of O'Reilly Media.) The most valuable benefit of providing open access to publications is it allows us to be "part of the conversation" in vibrant and productive online forums. ACM publications are widely cited; I just checked my own articles and blogs over the past few years and found I referred to *Communications*

16 times. Were the articles easier to search, read, and link to on the Web, they would play an even more important role in online discussions, as they do in professional publications. Reader comments would further enhance the value of the content.

However, this would not solve Vardi's concern over the cost of editing and publishing. He wrote that the prices charged for ACM journals not only cover the cost of their publication (an impressive achievement in itself) but yield a surplus that supports other ACM activities (truly commendable). I don't blame ACM for sticking to its partly closed model.

An alternative, if ACM were to go open, would be to subsidize publications through increased dues or other charges. I'm sure it would alter the calculation for ACM members (particularly students) when deciding whether to join for the first time or renew their memberships each year and might require new forms of fundraising. It's certainly common for organizations to ask members and donors to pay for development of information otherwise offered free to the world. If these publications represent the core offering to ACM members, the strategy is risky.

Andy Oram, Cambridge, MA

I'd like to propose yet another business model for the ACM Digital Library that blends both sides of the open vs. closed access debate, per Moshe Y. Vardi (July 2009). I agree that high-quality science publishing bears unavoidable costs even for electronic-only journals readers should pay for. But scientific papers should be shared free of charge with the largest audience possible. As a teacher, I must often explain basic algorithms and data structures by citing original pioneering papers. Because these "old" papers are still under copyright, students cannot freely and securely access them in digital libraries. They get only a limited view of their full technical coverage, getting the main aspects of highly cited papers while missing the full scope of the techniques being covered.

ACM should consider making available (for free) a set of highly rated CS papers from the Digital Library. This would offer students the historical papers that forged the science in the first

place, letting them in turn explore the functionalities of the Digital Library and inspiring their future interest in being subscribers. One way to do so might be to offer a free Education Library as a subset of the Digital Library.

Frank Nielsen, Paris, France

ACM Responds:

Concerning Stone's comments, please know that ACM does offer online-only subscriptions; a library need not buy a print package in order to subscribe to the Digital Library. In fact, most ACM library customers today belong to consortia for which the basic package is online-only access to the Digital Library with one free print package included; additional print packages are available to every member of a consortium, though few opt to buy them. Meanwhile, print long ago ceased to be ACM's "main profit center"; revenue from digital offerings far exceeds revenue from print. In any case, it is increasingly difficult to make a financial case for continuing print. For a number of publications, the print side must be subsidized, mainly to satisfy a dwindling set of library customers. Finally, ACM does offer electronic-only subscriptions to individual titles. Members and non-members alike are able to buy print-only subscriptions, electronic-only subscriptions, or print+electronic subscriptions to individual titles.

Bernard Rous, ACM Electronic Publishing Program Director, New York

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.

© 2009 ACM 0001-0782/09/0900 \$10.00

Coming Next Month in COMMUNICATIONS

*A View of the Parallel
Computing Landscape*

*An Interview with
David E. Shaw*

*Probing the Biomolecular
Landscape*

Smoothed Analysis

Plus, the latest news in shape-shifting devices, e-health records, and supercomputing, data management, and analysis.

In the Virtual Extension

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

Ballot Box Communication in Online Communities

Mu Xia, Yun Huang, Wenjing Duan, and Andrew B. Whinston

User interaction in online communities is one of the most noted features in the Web 2.0 era. A variety of sites devoted to sharing pictures (Flickr), video (YouTube), collective music recommendations (last.fm), and even voting for news articles that deserve attention (Digg), as well as social bookmarking (del.icio.us) have, for the first time, opened the door for users to interact with each other through short messages and other types of interaction. Nonmessage-based interactions have become a major force behind successful online communities. Recognition of this new type of user participation is crucial to understanding the dynamics of online social communities and community monetization.

Examining User Involvement in Continuous Software Development

Achita (Mi) Muthitacharoen and Khawaja A. Saeed

This study examines different factors that help promote users' participation in sending error reports through error report systems (ERS) that take a proactive approach by allowing users to send error-related information directly to the software firms when their software experiences a mishap. A survey conducted on 317 users and ERS factors were ranked according to their impacts on user's intention to send error report. Among several findings, the results reveal that value compatibility is the most influential factor. The study also discovered initial evidence of user's reflexive behavior in their interaction with the ERS.

Constructive Function-based Modeling in Multilevel Education

Alexander Pasko and Valery Adzhiev

The authors describe how a shape modeling and rendering framework based on the rapidly progressing function representation is used in the spirit of the educational constructionism theory to implement an active, creative, and collaborative learning process. The modeling language and software are being developed within an international HyperFun Project. The authors applied the theoretical framework and software tools on different levels of education starting from elementary schools

to doctoral thesis research in various areas related to mathematics, computer graphics, programming languages, artistic design, animation, and digital fabrication. They illustrate the presented approach by practical experience examples from different educational institutions and countries.

One Size Does Not Fit All: Legal Protection for Non-Copyrightable Data

Hongwei Zhu and Stuart E. Madnick

The Web is the largest data repository on earth and Tim Berners-Lee has noted "the exciting thing is serendipitous reuse of data: one person puts data up there for one thing, and another person uses it another way." However, data reuse faces certain legal challenges. As computing professionals develop new Web technologies, we must understand the legal implications of using them for data reuse purposes. After reviewing legal and policy issues, the authors discuss a framework for policies that maximally allow value-creating data reuse without diminishing the incentives of compiling databases and making them available on the Web.

The State of Corporate Web Site Accessibility

Eleanor T. Loiacono, Nicholas C. Romano, Jr., and Scott McCoy

Web accessibility continues to have important social, legal, and economic implications for e-commerce. Over 50 million Americans and around 600 million people worldwide possess some sort of disability. In this study, the authors expound on a previous *Communications* article that surveyed Fortune 100 Web sites for their level of accessibility at a snapshot in time. This study adds three additional data sets for a total of four—2000, 2002, 2004, and 2005—to present a longitudinal perspective. The authors examine the reasons why global companies should care about accessibility and offer recommendations on how to get started.

Reducing Employee Computer Crime through Situational Crime Prevention

Robert Willison and Mikko Siponen

Employee computer crime represents a substantial threat for organizations.

Yet information security researchers and practitioners currently lack a clear understanding of how these crimes are perpetrated, which consequently hinders security efforts. The authors argue that recent developments in criminology can help to address the insider threat. More specifically, they demonstrate how an approach, entitled Situational Crime Prevention, can not only enhance an understanding of employee computer crime, but also strengthen security practices designed to address this problem.

Modified Agile Practices for Outsourced Software Projects

Dinesh Batra

In recent years, agile practices have become popular in the software development industry. However, some agile practices break down when faced with the realities of outsourced development, including the larger size of the typical project, and the geographical, language, temporal, social, and cultural barriers. This article explores how agile practices must be reevaluated in the broader software development environment.

Technical Opinion: Falling into the Net: Main Street America Playing Games and Making Friends Online

James Katz and Ronald E. Rice

Findings from a U.S. survey of the general population identify how the Internet is affecting the daily lives of ordinary people. A nationally representative random survey of 1,404 people finds that, on balance, there is almost no evidence to support the harsh contentions that the Internet is harmful or breeds sad, lonely people as has been asserted. Neither is there evidence to indicate the Internet is male-dominated. Rather, the survey findings indicate that millions of people find community online, and many new friendships have been forged. In fact, a significant fraction of those friendships have extended from the virtual to the face-to-face world. So rather than people "dropping out" of life to become hermits, data shows the Net is a pro-social medium, resource, and network that brings people together.

ACM Digital Library

www.acm.org/dl

The Ultimate Online INFORMATION TECHNOLOGY Resource!



Powerful and vast in scope, the **ACM Digital Library** is the ultimate online resource offering unlimited access and value!

The **ACM Digital Library** interface includes:

- **The ACM Digital Library** offers over 40 publications including all ACM journals, magazines, and conference proceedings, plus vast archives, representing over 2 million pages of text. The ACM DL includes full-text articles from all ACM publications dating back to the 1950s, as well as third-party content with selected archives. **PLUS NEW: Author Profile Pages with citation and usage counts and New Guided Navigation search functionality!**

www.acm.org/dl

- **The Guide to Computing Literature** offers an enormous bank of over one million bibliographic citations extending far beyond ACM's proprietary literature, covering all types of works in computing such as journals, proceedings, books, technical reports, and theses! www.acm.org/guide

- **The Online Computing Reviews Service** includes reviews by computing experts, providing timely commentary and critiques of the most essential books and articles.

Available only to ACM Members.

Join ACM online at www.acm.org/joinacm

To join ACM and/or subscribe to the Digital Library, contact ACM:

Phone: 1.800.342.6626 (U.S. and Canada)

+1.212.626.0500 (Global)

Fax: +1.212.944.1318

Hours: 8:30 a.m.-4:30 p.m., Eastern Time

Email: acmhelp@acm.org

Join URL: www.acm.org/joinacm

Mail: ACM Member Services

General Post Office

PO Box 30777

New York, NY 10087-0777 USA

*Guide access is included with Professional, Student and SIG membership. ACM Professional Members can add the full ACM Digital Library for only \$99 (USD). Student Portal Package membership includes the Digital Library. Institutional, Corporate, and Consortia Packages are also available.



Association for
Computing Machinery

Advancing Computing as a Science & Profession

The *Communications* Web site, <http://cacm.acm.org>, features 13 bloggers in the BLOG@CACM community. In each issue of *Communications*, we'll publish excerpts from selected posts, plus readers' comments.

DOI:10.1145/1562164.1562169

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

Saying Good-bye to DBMSs, Designing Effective Interfaces

Michael Stonebraker discusses the problems with relational database management systems and possible solutions, and Jason Hong writes about interfaces and usable privacy and security.



From Michael Stonebraker's "The End of a DBMS Era (Might be Upon Us)"

Relational database management systems (DBMSs) have been remarkably successful in capturing the DBMS marketplace. To a first approximation they are "the only game in town," and the major vendors (IBM, Oracle, and Microsoft) enjoy an overwhelming market share. They are selling "one size fits all"; i.e., a single relational engine appropriate for all DBMS needs. Moreover, the code line from all of the major vendors is quite elderly, in all cases dating from the 1980s. Hence, the major vendors sell software that is a quarter century old, and has been extended and morphed to meet today's needs. In my opinion, these legacy systems are at the end of their useful life. They deserve to be sent to the "home for tired software."

Here's why.

If we examine the nontrivial-sized DBMS markets, it turns out that cur-

rent relational DBMSs can be beaten by approximately a factor of 50 in most any market I can think of. What follows are a few examples.

In the data warehouse market, a column store beats a row store by approximately a factor of 50 on typical business intelligence queries. The reason is because column stores read only the columns of interest to the query and not all of them. In addition, compression is more effective in a column store. Since the legacy systems are all row stores, they are vulnerable to competition from the newer column stores.

In the online transaction processing (OLTP) market, a lightweight main memory DBMS beats a row store by a factor of 50. Leveraging main memory and the fact that no DBMS application will send a message to a human user in the middle of a transaction allows an OLTP DBMS to run transactions to completion with no resource contention or locking overhead.

In the science DBMS market, us-

ers have never liked relational DBMSs and want a non-relational model and query facility. (This was the topic of my last CACM blog, "DBMSs for Science Applications: A Possible Solution.")

If you are storing Resource Description Framework (RDF) data, which is popular in the bio community and elsewhere, then column stores are very good at certain RDF workloads. In addition, other ideas, such as RDF-3X, will beat conventional DBMSs in other situations. Lastly, native RDF engines (e.g., Virtuoso, Sesame, and Jena) may well gain traction. The point is that something else will beat conventional row stores in this market.

Text applications have never used relational DBMSs. This was pointed out to me most clearly by Eric Brewer nearly 15 years ago in the early days of Inktomi. He wanted to use a relational DBMS to store the results of Web crawling, but found relational DBMSs to be two orders of magnitude slower than a home-brew system. All the major Web-search engines use home-brew text software to serve us search results. None use relational DBMSs.

Even in XML, where the current major vendors have spent a great deal of energy extending their engines, it is claimed that specialized engines, such as Mark Logic or Tamino, run circles around the major vendors, according to a private communication by Dave Kellogg.

In summary, one can leverage at least the following ideas to get superior performance:

A non-relational data model. If the

user's data is naturally something other than tables and if simulating his natural data model on top of tables is awkward, then chances are that a native implementation of the natural data model will significantly outperform a conventional relational DBMS. This is certainly true in scientific data.

A different implementation of tables. If something other than a row store accelerates the user's queries, then a direct implementation of the relational model using non-row store technology will run circles around a conventional relational DBMS. This is true in the data warehouse marketplace.

A different implementation of transactions. Current row stores give you a "one size fits all" implementation of transactions. This can be radically beaten if a user has lesser requirements or if the system can take advantage of workload-specific features. This is true in the OLTP marketplace.

One of these characteristics is true in every market I can think of. Hence, in my opinion, the days of a "one size fits all" monolithic DBMS are at an end. The replacement will be a collection of vertical market-specific engines, with much higher performance.

You might ask, "What if I don't care about performance?" The answer: Run one of the open source relational DBMSs. They are mature, reliable, and, best of all, free.

You might also ask, "I am dug in deep with my current vendor(s). What do I do?" The answer: Take some portion of your DBMS budget and allocate it to new solutions. Over time, you will move onto better technology.

Reader's comment

It is very true that relational DBMSs are overhyped for not so valid reasons. The current trends also showcase that there are viable alternatives to relational DBMSs, which can beat them at their own game. Also, the emergence of distributed key-value stores, such as Cassandra and Voldemort, proves the efficiency and cost effectiveness of their approaches.

Also, the recently concluded NoSQL conference discussed at length how distributed, non-relational databases work, along with overview of the emerging alternatives in this space.

—Pavan Yara



From Jason Hong's "Designing Effective Interfaces for Usable Privacy and Security"

I often cringe when I hear highly technical engineers talk about people.

I usually hear broad generalizations tossed about, like "people are lazy, that's why they can't use the system" or "people don't understand security." The worst is "people are just stupid."

With this kind of attitude, it's no surprise there are so many complicated user interfaces in the world, let alone in privacy and security. Failing to try to understand things from the user's point of view is the cardinal sin in user interface design.

With this in mind, I thought it would be good to shift focus in this blog entry away from individual case studies of usable privacy and security, and look at the bigger picture of how to design better user interfaces.

Now, how to craft an effective user interface is a very involved topic that one can study for years, and there are lots of great Web sites and books out there. Effective user interface design combines our understanding of aesthetics, technology, and human behavior to develop artifacts that are useful, usable, and desirable for a specific target audience.

What makes usable privacy and security different from designing other interfaces is that privacy and security are often secondary tasks. People don't go to an e-commerce site explicitly wanting to protect their credit cards and email addresses; they go there to buy things. Security and privacy are obvious things they want while accomplishing their main goal, in the same manner that they want the Web site to also be fast and usable.

Roughly, there are three broad strategies for usable privacy and security (note that these aren't mutually exclusive):

- ▶ make the interface invisible
- ▶ make the interface more understandable
- ▶ train the users

A good example of better security by making the interface invisible is Secure Sockets Layer. End users don't have to do anything special, and all

of their network traffic is transparently encrypted.

Oftentimes, we just need to make the user interface more understandable to end users. This might be accomplished through better layout, simplified task flows, better visualizations, or more appropriate metaphors (why do we sign digital documents using keys, anyway?).

Finally, some user interfaces may also require training the users. One common misconception about user interfaces is that they should be "intuitive" (a description that always raises a red flag with me). If you're a Star Trek fan like I am, you might remember that famous scene in *Star Trek IV* where Montgomery Scott, the ship's engineer, tries to use a Macintosh computer. After attempting to talk to the computer and getting no response, he picks up the mouse and tries talking into it. Intuitive indeed.

Applications are always designed for a specific context, for specific purposes, and for a specific target audience. The best designs will empower people and let them get started quickly, while also providing a way for them to get better.

As such, some applications will require some level of training. The training might range from a basic understanding of how to zoom in and out on the iPhone (which Apple cleverly trained people how to do, with their television ads), all the way to learning how to drive a car (something we actually start training our children to do since birth, given how ingrained cars are in society).

Now, this doesn't mean that you can get away with a disastrous user interface and expect people to have to train how to use it, but it also doesn't mean that all user interfaces should be walk up and use either. You have to balance ease-of-use with power and flexibility *for your specific audience and your specific goals*. As Silicon Valley pioneer Doug Engelbart once noted, if ease of use was all that mattered, we'd all still be riding tricycles. □

Michael Stonebraker is an adjunct professor at the Massachusetts Institute of Technology. Jason Hong is an assistant professor at Carnegie Mellon University.



DOI:10.1145/1562164.1562170

David Roman

What You Read on Your Summer Vacation

As publishers, it is imperative we always stay attuned to the kind of editorial material our audience finds most professionally valuable and engaging. Indeed, we devour Web analytics about *Communications'* site to better serve our current audience and to draw more into the fold. And we intend to share that information with you on a regular basis. Here, we present the most popular articles and sections this past summer (starting with the Memorial Day weekend, May 22) as indicated by our latest site statistics.

Top Articles

FYI: Only full-text articles are ranked, though some abstracts got more pageviews.

- | | |
|--|--|
| 1. The Five-Minute Rule 20 Years Later
cacm.acm.org/magazines/2009/7/32091 | 6. Award-Winning Paper Reveals Key to Netflix Prize
cacm.acm.org/news/32450 |
| 2. One Laptop Per Child: Vision vs. Reality
cacm.acm.org/magazines/2009/6/28497 | 7. Time for Computer Science to Grow Up
cacm.acm.org/magazines/2009/8/34492 |
| 3. Whither Sockets?
cacm.acm.org/magazines/2009/6/28495 | 8. How Computer Science Serves the Developing World
cacm.acm.org/magazines/2009/6/28498 |
| 4. CS Education in the U.S.: Heading in the Wrong Direction?
cacm.acm.org/magazines/2009/7/32090 | 9. Why 'Open Source' Misses the Point of Free Software
cacm.acm.org/magazines/2009/6/28491 |
| 5. API Design Matters
cacm.acm.org/magazines/2009/5/24646 | 10. Conferences vs. Journals in Computing Research
cacm.acm.org/magazines/2009/5/24632 |

Top Blog Posts

FYI: Michael Stonebraker's #1 entry had more than 10 times the traffic of his #5 entry.

- | |
|--|
| 1. The End of a DBMS Era (Might Be Upon Us)
cacm.acm.org/blogs/blog-cacm/32212 |
| 2. The Siren Song of Startups
cacm.acm.org/blogs/blog-cacm/29807 |
| 3. The Biggest Gains Come From Knowing Your Data
cacm.acm.org/blogs/blog-cacm/33805 |
| 4. What Is a Good Recommendation Algorithm?
cacm.acm.org/blogs/blog-cacm/22925 |
| 5. DBMSs for Science Applications: A Possible Solution
cacm.acm.org/blogs/blog-cacm/22489 |

Top Sections:

FYI: The homepage got more pageviews than all of these sections combined.

- | |
|--|
| 1. Magazine Archive: cacm.acm.org/magazines |
| 2. BLOG@CACM: cacm.acm.org/blogs/blog-cacm |
| 3. News: cacm.acm.org/news |
| 4. Blogs: cacm.acm.org/blogs |
| 5. Author Guidelines: cacm.acm.org/about-communications/author-center |



Top Issues:

FYI: The site's momentum is evident.

- | |
|---|
| 1. June 2009: cacm.acm.org/magazines/2009/6 |
| 2. July 2009: cacm.acm.org/magazines/2009/7 |
| 3. May 2009: cacm.acm.org/magazines/2009/5 |
| 4. April 2009: cacm.acm.org/magazines/2009/4 |
| 5. March 2009: cacm.acm.org/magazines/2009/3 |

ACM Member News

NEW IMAGE FOR COMPUTING INITIATIVE

An interim report by ACM and the WGBH Educational Foundation as part of a project to improve the image of computer science among high school students confirms a significant gender gap among college-bound students in their opinions about computing as a college major or career. Funded by the National Science Foundation, the research found that 74% of boys, regardless of race or ethnicity, reported that a college major in computer science was a "very good" or "good" choice for them. However, only 32% of girls rated it as a "very good" or "good" choice. The ACM-WGBH Educational Foundation report, which covers the first phase of the New Image for Computing initiative, seeks to answer why interest in computer science in U.S. colleges and pursuing computer-related careers is declining.

"We know that the number of computer science majors is not meeting projected work force needs," noted John White, ACM CEO and co-principal investigator for the project. "Many factors contribute to the low interest in computer science, but the image of the field is a key element in current perceptions among this population."

The gender gap extended to computer science as a potential career choice as well as a field of study. From a selection of 15 possible careers, computer science came in fourth among the respondents, with 46% rating it "very good" or "good." However, while 67% of all boys rated computer science highly as a career choice, only 26% of girls rated it "very good" or "good."

The research showed little racial or ethnic differentiation in young people's attitudes toward computer science, with it being held in high regard by college-bound African American and Hispanic boys, but these two groups remain underrepresented in both academia and the computer science work force.

Entering a Parallel Universe

The multicore processors that help extend Moore's Law may run afoul of Amdahl's Law.

FOR YEARS, SOFTWARE programmers had it easy when they wanted to write faster and more feature-laden applications. They just did it, thanks to the seemingly eternal verity of Moore's Law, which states that the density of transistors on a chip will double every two years. The size of transistors has continued to shrink, making it possible for more transistors to be placed on a die of the same size, but clock speeds haven't increased, due to thermal issues. Intel and other chip manufacturers have opted to use the extra transistors to build multiple processor cores on the same die, as opposed to increasing the functionality or sophistication of each individual core. In order to take advantage of these multiple cores, applications must be rewritten to accomplish their task, using multiple parallel threads of execution.

Parallel programming is not new; it has been a mainstay in high-performance computing for decades. However, the vast majority of general-purpose platforms have been designed to operate sequential applications, in which one instruction logically follows its predecessor to accomplish a given task. In parallel programming,

numerous calculations are performed simultaneously, operating on the principle that large problems can often be divided into many smaller ones, which are then solved concurrently. However, generations of programmers for mainstream platforms have never had to work in parallel.

"We have come to parallel programming not because of the success of our software, but because of the failure of our hardware," says Tim Mattson, a senior research scientist at Intel. "If the hardware can't give it to you with a single thread anymore you have to figure out how to do parallel. So it's kind of an

urgent situation here. We have to crack this one." (See "Face the Inevitable, Embrace Parallelism," on p. 36.)

What to Parallelize

There are numerous "yes, but..." scenarios inherent in bringing parallel programming to mainstream machines and developers. For instance, many programming experts say the return on investment for changing sequential code for the next generation of dual-core or quad-core machines might not be large enough to make the transition worthwhile. As a result, researchers and toolmakers have a short grace period in which to examine exactly which applications are worth parallelizing.

"The consequence of having to move to multicore is that we have to figure out how to use parallelism in places where we need more performance," says Jim Larus, director of software architecture for cloud computing futures at Microsoft. "Not ev-



Students participating in a class on multicore programming at the University of Illinois at Urbana-Champaign's Universal Parallel Computing Research Center in June 2009.

everything needs to be parallel. What's it going to do for you to make Word run a little faster?"

Larus says some applications, such as speech recognition, for which parallel programming is seen as a requirement, might benefit more from algorithmic improvements of existing serially written applications instead of converting to parallel processing. "The people working on [speech recognition] at Microsoft say a machine that's 10 times faster would probably reduce the error rate by a few tenths of a percent," says Larus. "They see the future in terms of better algorithms, not more computation. We're saying we can keep giving you exponential growth in compute power for certain types of programs, and people are telling us 'That's not really what we need for what we're doing' or 'That's not enough for what we're doing.'"

Larus's cautions aside, the computer industry is moving en masse to multicore machines, and users will expect to receive additional performance for their money—performance that will often depend on parallel applications. Therefore, some experts say, there is a danger in not immediately starting to train programmers on the requirements of parallel programming. This

consciousness raising is important at all levels, from industry veterans to undergraduate students. "Sequential programming models do not work well enough," says Maurice Herlihy, a professor of computer science at Brown University. "We can more or less keep busy" four cores or fewer, he says, "but beyond that we'll have to rethink some things. If you can't deliver more value with more cores, why would anybody ever upgrade?" Herlihy sees a peril that the engine of progress that has driven computer science for decades could run out of fuel, with dire consequences. "If this were to dissipate, then all the smart students would go to bioengineering or something, and computer science as a field would suffer." Indeed, he says, "even one generation of stagnation could do lasting damage."

Incremental Integration

Computer scientists on university faculties say academia is debating how and when to introduce parallel programming throughout the curriculum, instead of just offering an upper-level course as is now common. Both Brown's Herlihy and Guy Blelloch, a professor of computer science at Carnegie Mellon University, say the early emphasis should be on higher-level

parallel concepts and not on coding particulars such as languages or development frameworks.

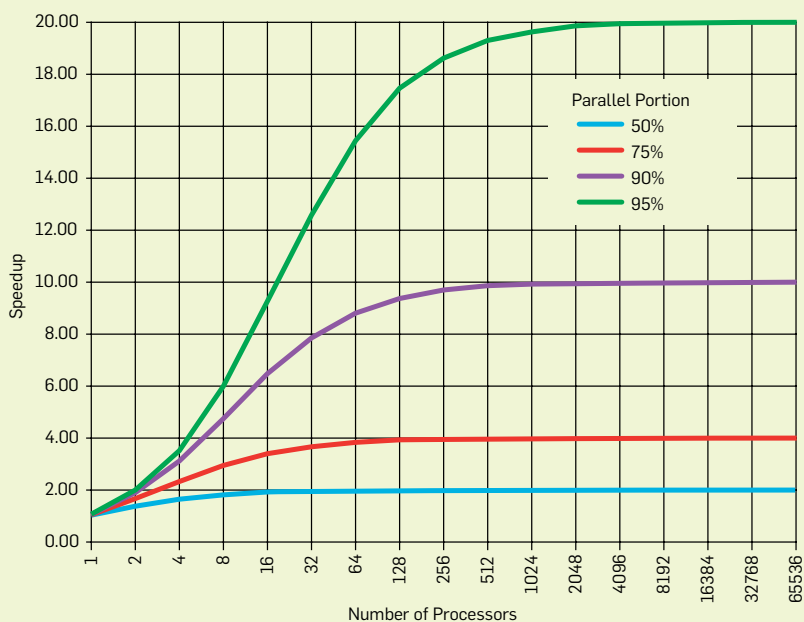
Yet without some sort of introduction to the emerging post-collegiate parallel programming practice and tools environment, these new engineers might need even more training. Herlihy says that existing parallel frameworks—such as OpenMP, which dates to 1997, and the newly released OpenCL—are well suited for professional programmers, but not for students, who largely program in Java. This lack of grounding in the fundamentals of parallel code writing could lead to a looming disconnect between what new programmers know and what industry needs.

Intel's Mattson, who worked on both frameworks, says one of the major blind spots of both OpenMP and OpenCL is a lack of support for managed languages such as Java. He also says the idea that there may be some type of revolutionary parallel programming language or approach on the near-term horizon that solves the multicore conundrum is misplaced. "Programmers insist on incremental approaches," Mattson says. "They have a huge base of code written in established languages they will not throw away to adopt a whole new language, and they have to be able to incrementally evolve this legacy of code into the parallel space."

The good news is that tools to assist programmers in this task of incrementally parallelizing code are proliferating on the market. Examples include Cilk++ from Cilk Arts, a Burlington, MA, company that extends the work of the Cilk Project at MIT. Cilk++ allows parallel programs written in C++ to retain serial semantics, which in turn permits programmers to use serial methodologies. CriticalBlue, an Edinburgh, Scotland-based company, recently released Prism, a parallel analysis and coding tool that CEO David Stewart says works with C or C++ and that allows users to explore parallelization strategies—which pieces to run in parallel, which dependencies to break, how many cores to use, and so on—before touching the code.

The most sensible way to implement parallelism, Stewart contends, is to enable software developers to analyze how much potential parallelism is in their code and to determine the min-

Amdahl's Law



Named after computer architect Gene Amdahl, Amdahl's Law is frequently used in parallel programming to predict the theoretical maximum speedup using multiple processors.

Academia is debating when and how to add parallel programming to the curriculum, instead of only offering an upper-level course as is now common.

imum set of code changes required to exploit that parallelism. “Put another way,” Stewart says, “how much does Amdahl’s Law screw me up?”

A Firm Ceiling

For years, the promise of parallel computing has run afoul of the harsh reality of an axiom posited by computer architect Gene Amdahl in 1967. Amdahl’s Law puts a firm ceiling on the benefit of converting code from sequential to parallel. It states that the speedup of an application using multiple processors in parallel computing is limited by the time needed for the sequential fraction of the program. The upshot: going down the path of parallelism will not necessarily reap rewards.

“If 50% of your program is serial and the other half can be parallelized, the biggest speedup you’re going to see is a factor of two,” says Microsoft’s Larus. “It doesn’t matter how many cores you have. And that doesn’t seem very compelling if you’re going to have to rewrite a huge amount of software.” Amdahl’s Law, Larus says, might very well mean that a wholesale rush to convert serial applications to parallel platforms in order to preserve a Moore’s Law pace of progress would be misguided. In many cases, it will be more cost-effective to improve serial applications’ performance via algorithmic advances and custom circuitry rather than going for the marginal return on investment that parallelizing those applications might provide.

For Larus, reconciling the true computational needs of future applications with the overall move to parallel-capable

multicore processors must entail rigorous evaluation of what type of application might deliver the most benefit to users. Because the bulk of general-purpose computing has been done successfully on serial platforms, he says it’s been difficult to pin down exactly which applications might derive the most benefit from parallelization.

“This is a challenging problem,” he says. “A lot of people take the attitude, ‘If we build it they will come,’ and that may very well happen—there might be a killer app. But not knowing what that is makes it really hard to build the infrastructure and the tools to facilitate the app.”

So far, Larus says, the computer engineering community is basing its idea of what future general-purpose multicore platforms are capable of due to niche applications such as high-performance scientific data analysis software, where parallelization has shown its value. But there’s no guarantee it will be possible to extrapolate from this experience to create the development frameworks most programmers will need as parallelism goes mainstream in general-purpose computing.

“In this case we’re going backwards; we’re building the tools based on our experience with high-performance computing or scientific computing and saying people are going to need this. And that may be true, but it has a funny feel to me—to have the tools leading.”

Larus says the key to successful parallel platforms might be in finding a way to combine existing discrete serial platforms. One example, he says, might be a virtual receptionist that needs to process visual cues from a camera taking images of a visitor while also responding to spoken queries such as the visitor’s request for directions to a nearby restaurant. “This type of thing has been gradually building up in discrete fields over the years,” Larus says. “When you put it together there are all these big, independent pieces that only interact at these well-defined boundaries. That’s a problem that’s actually easy to parallelize.”

Gregory Goth is an Oakville, CT-based writer who specializes in science and technology. David A. Patterson, University of California, Berkeley, contributed to the development of this article.

© 2009 ACM 0001-0782/09/0900 \$10.00

Milestones

Computer Science Awards

President Barack Obama and the National Science Foundation (NSF) recently honored members of the computer science community for their innovative research. Among them:

NSF CAREER AWARD



Tiffany Barnes, an assistant professor in the department of computer science at the University of North Carolina at Charlotte, has received a Career Award from the NSF for her research on artificial intelligence and education. The goal of Barnes’ project is to create technology for a new generation of data-driven intelligent tutors, enabling the rapid creation of individualized instruction to support learning in science, technology, engineering, and mathematics fields.

PRESIDENTIAL EARLY CAREER AWARDS

President Obama has named 100 beginning researchers as recipients of the Presidential Early Career Awards for Scientists and Engineers (PECASE), the highest honor bestowed by the U.S. government on young professionals in the early stages of their independent research careers.

Of the 100 PECASE winners, 15 are computer scientists. They are: Cecilia R. Aragon, Lawrence Berkeley National Laboratory; David P. Arnold, University of Florida; Seth R. Bank, University of Texas, Austin; Joel L. Dawson, Massachusetts Institute of Technology; Chris L. Dwyer, Duke University; Anthony Grbic, University of Michigan; Carlos E. Guestrin, Carnegie Mellon University; Sean Hallgren, Penn State University; Yu Huang, University of California, Los Angeles; Gregory H. Huff, Texas A&M University; Sanjay Kumar, University of California, Berkeley; Rada F. Mihalcea, University of North Texas; Adam D. Smith, Penn State University; Adrienne D. Stiff-Roberts, Duke University; and Sharon M. Weiss, Vanderbilt University.

Medical Nanobots

Researchers working in medical nanorobotics are creating technologies that could lead to novel health-care applications, such as new ways of accessing areas of the human body that would otherwise be unreachable without invasive surgery.

SINCE KAREL CAPEK first used the word “robot” in print in a 1920 play, a vast array of autonomous electro-mechanical systems have emerged from research labs, making their way onto production lines for industrial tasks, into toy stores for entertainment, and even into homes to perform simple household jobs. While the bulk of robotics research strives to make robots more useful and more capable of even greater levels of autonomy, several labs are attempting to make robotic systems much smaller. One of the most active areas of such research is medical nanorobotics, an emerging field positioned at the intersection of several sciences.

As a discipline, medical nanorobotics remains young for now, but many scientists are already demonstrating new developments they say will form the foundations for the next major breakthroughs in this area. Such breakthroughs could lead to novel applications that offer new ways of accessing small spaces in the human body that would otherwise be unreachable without invasive surgery.

“Nanorobotics can play a major role in medical applications, especially for target interventions into the human body through the vascular network,” says Sylvain Martel, director of the nanorobotics laboratory at École Polytechnique de Montréal. “In many types of interventions, medical specialists are lacking appropriate tools to do a good job, and I believe that nanorobotics could bring new methods and tools to these particular applications.”

Recent fabrication, actuation, and steering demonstrations of nanoscale robots represent the first crucial steps toward developing real-world applications for targeted drug delivery and other uses. But researchers say that with

many engineering and medical challenges remaining to be met, clinically usable medical nanobots might be viable only after several more years of work in this area. “I believe that the first real application that will have a huge impact is in targeted cancer therapy, such as delivering therapeutic agents directly to the tumor through the vascular network,” says Martel.

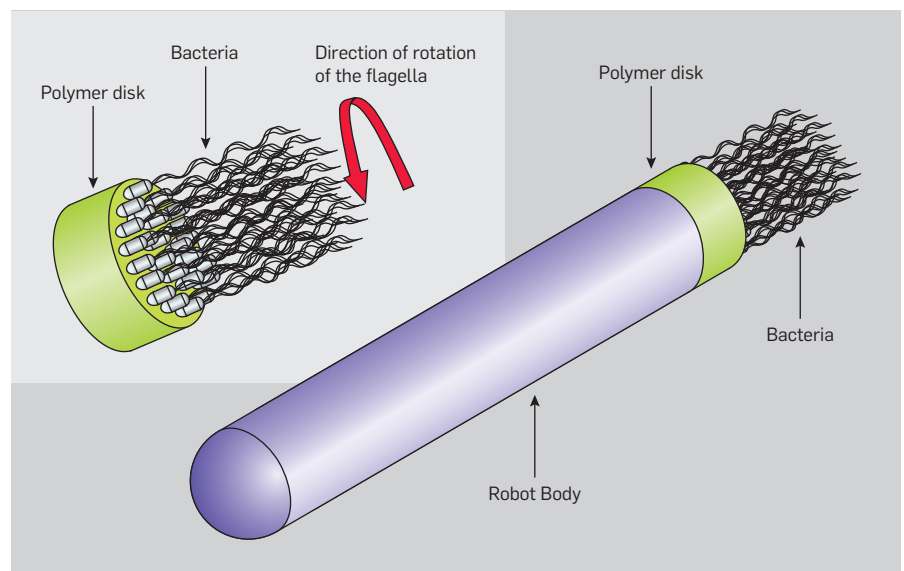
Currently, Martel and his team are focused on developing a medical application designed to target regions inaccessible to traditional catheterization techniques. The platform they created uses magnetic resonance imaging (MRI) for feeding information to a controller that is responsible for steering the nanobots along blood vessels. The nanobots, which consist of magnetic carriers and flagellated bacteria that can be controlled by computer and loaded with therapeutic and sensing agents, essentially serve as wireless robotic arms that can perform remote tasks.

“Unlike known magnetic targeting

methods, the present platform allows us to reach locations deep in the human body using real-time control,” Martel says. Still, he predicts it will take three to five years before the system reaches maturity, meaning complete computer-based control of the propulsion and steering mechanisms.

Another researcher designing a similar approach to controlling nanobots is Metin Sitti, director of the nanorobotics lab at Carnegie Mellon University. Sitti and his team are working on building nanobots for drug-delivery applications. In one recent project, he and his team have used bacteria to move nanoscale robots, which use the chemical energy inside the bacteria and in the environment for propulsion. In addition to this propulsion method, Sitti and his team have experimented with optical and magnetic stimuli to coax the bacteria into decelerating, stopping, and moving again.

But as with other similar projects in this area, Sitti and his team are facing



A nanobot created at Carnegie Mellon University and demonstrated to be functional in real-world experiments. The flagella motion of the bacteria's cells propel the nanobot, which is controlled by the application of environmental stimuli.

several fundamental challenges. “Such bacteria-propelled nanobots are limited by the stochastic nature of cellular motion, and by the relatively brief lifetimes of bacteria,” he says. In addition, Sitti says he and his team must develop more effective ways to communicate with nanobots once they are inside a body. “Methods must be found to program and control large numbers of nanobots,” Sitti says. “This will be necessary if such devices are to treat large areas of the body, to increase the speed and success of medical operations, and to deliver sufficient amounts of drugs to their targets.”

Scientists working in this area say the nanorobotic systems developed by Martel, Sitti, and other researchers could lead to new surgical techniques far more sophisticated and far less invasive than methods currently in use. Such techniques would rely on devices capable of entering the human body through natural orifices or very small incisions to perform diagnostic procedures or repair tissue. “The mechanisms of life operate at the nanoscale,” says Aristides Requicha, director of the laboratory for molecular robotics at the University of Southern California. “If we build devices at their scale, we will be able to interact intimately with them.”

One goal of Requicha’s work in this area is to overturn the basic paradigm of today’s medicine, and to shift from a treatment model to a prevention model through the use of in-body sensors that check for and kill pathogens before the patient has any symptoms. Essentially, Requicha’s vision entails rethinking the traditional sequence of patients demonstrating symptoms and then seeking medical care for their ills. “In the long run,” he says, “I would like to build artificial and preferably programmable cells.” In the meantime, though, one project Requicha and his team are working on is a network of wireless nanosensors capable of operating in biological environments. “This network would give us unprecedented capabilities to study cell biology by being able to acquire data in real time and for extended periods,” he says.

Near-Term Applications

While some research in this field remains theoretical and might never directly lead to real-world applications,

Aristides Requicha's research aims for a preventive health model in which in-body sensors check for and kill pathogens before a patient exhibits any symptoms.

several nanorobotics labs focus specifically on projects that might have near-term practical applications. “One aspect of entering these fields that was particularly important to me, as an engineer, was to make sure there were genuine applications on the horizon that made sense,” says Bradley Nelson, head of the institute of robotics and intelligent systems at ETH Zürich. “It rapidly became clear that applications in biological research were possible, but then it became even more clear that the potential in medicine was the real reason for pursuing these fields.”

Among many projects, Nelson’s group is creating artificial flagella designed to mimic natural bacteria in both size and swimming technique and is working on nanobots for retinal surgery. The challenges he and his team face, as with other researchers working in this area, are numerous. Still, Nelson says he remains optimistic, and points to a recent spinoff called FemtoTools, in Zürich, Switzerland, that is already marketing micromanipulation products, such as force sensors and micro-grippers. “With sufficient resources and energy and the backing of doctors and business people,” he says, “retinal therapies using nanobots will be possible within five years.”

With nanorobotics labs working hard to address fundamental issues in physics, biology, and computer science as they seek to create viable medical applications, at least one major challenge resides on a more social level. One of the most frequently cited difficulties of working in this field is the

interdisciplinary nature of the research itself, which requires not only combining advanced science in health with advanced science in robotics, but also the ability to communicate in the language that medical professionals use.

Nelson’s group, for example, consists of roboticists, mechanical engineers, electrical engineers, software engineers, computer vision researchers, materials scientists, and chemists. And the team works directly with doctors and biologists. “Trying to understand what all these disciplines are about and how they can work together is a major challenge and, to me, one of the most stimulating aspects of this field,” Nelson says. Martel points to a similar experience. “In my office, I can talk about a new imaging algorithm on an MRI machine, and five minutes later have a conversation about microelectronic circuits, antibodies to connect nanoparticles to miniature robots, or genetics to enhance the molecular motor of flagellated bacteria,” he says.

Requicha, for his part, says interdisciplinary work is exciting but not easy. “This is an issue not only at the research level, but also educationally,” he says. “How do we prepare students to work in this field?”

In addition to the challenges associated with the interdisciplinary nature of the research, researchers cite safety issues, health concerns, and government regulation as other key issues. Swallowing or injecting miniature robots is not something many patients would readily agree to do without assurances of safety, or at least some demonstrable evidence that the potential benefits outweigh the possible risks. Because human physiology is complex, dynamic, and even different from person to person, reliably producing such evidence likely will remain an engineering challenge for years to come.

Despite the many challenges, researchers say the efforts will yield positive results in the end, with technology that revolutionizes medicine by making health care less expensive and less painful, and enabling medical professionals to target diseases for diagnosis and treatment, precisely and locally. ■

Based in Los Angeles, **Kirk L. Kroeker** is a freelance editor and writer specializing in science and technology.

© 2009 ACM 0001-0782/09/0900 \$10.00

Facing an Age-Old Problem

Researchers are addressing the computing challenges of older individuals, whose needs are different—and too often disregarded.

IT'S NO SECRET that computers and the Internet have changed society in ways that weren't imaginable only a quarter-century ago. The ability to connect with other people all over the globe, read about events as they unfold, shop online, and manage information has profoundly changed the landscape—and mind-set—of modern society. More than three-quarters of households in the U.S. have computers, and the numbers are exploding all over the world.

Today, it's difficult to imagine a world without computers. And while the so-called digital divide remains—the gulf between the affluent and poor in terms of computer accessibility—researchers are discovering that another important barrier exists. “Many older people face formidable challenges when it comes to using computers,” says Vicki Hanson, manager for the accessibility research group at IBM. “They are different from other segments of the population in terms of both cognitive and physical capabilities.”

The challenges of dealing with an aging world population haven't been lost on researchers, psychologists, and technology designers. Although computer and software manufacturers have made some strides in building easier-to-use systems—including specialized Web browsers, ergonomic mice and keyboards, and accessibility functions such as the ability to zoom and magnify text and graphics—there is still a long way to go to provide a computing environment that's ideal for older individuals.

“Technology creates a lot of potential in terms of enhancing the quality of life, independence, and well-being of older adults,” observes Sara J. Czaja, professor of psychiatry and behavioral sciences and director of the center on research and education for aging and technology at the University of Miami. “It opens up work and personal opportunities and allows older adults to stay socially connected. But this group has



Creating an ideal computing environment for older adults poses many challenges.

different needs and they're too often overlooked.”

Age Matters

As commerce, health care, government services, and work migrate online, the need to use computing devices is shifting from *desirable* to *essential*. According to Czaja, older individuals don't have any particular aversion

“When people of any age can't figure things out,” says Sara J. Czaja, “they tend to avoid the technology.”

to using computers and technology. “They are entirely receptive,” she says. So, where does the problem lie? “They often don't understand the benefits or they're unable to use the system easily,” she says. “When people of any age can't figure things out, they tend to avoid the technology.”

The challenges are growing. As computing expands from desktop and notebook systems to a wider universe of devices—including phones, home entertainment centers, navigation systems in cars, security systems, and high-tech climate controls—older adults are increasingly feeling overwhelmed and frustrated. “There are a lot of older people who have a lot of trouble with mobile phones. They simply can't use a typical phone because the interface is confusing and the buttons are too small,” Czaja explains.

Yet, poor vision, shaky fingers, and fading memory are only a few of the manifestations of older age. It's not unusual for individuals to experience

problems related to hearing, tactile perception, and the ability to recognize movement, notes Takashi Saito, manager of the accessibility center for IBM's Tokyo Research Lab. In fact, many of these older individuals have "multiple slight disabilities" that create a unique set of challenges. As a result, it's not simple to engineer a straightforward solution for a single problem.

For example, a person with failing eyesight might benefit from a "blind touch" keyboard, but deteriorating motor skills might make it a challenge to use any keyboard. Another older person might find it easier to use a specialized Web browser—or an alternative Web site—that simplifies layout and design elements, but she may still have problems figuring out what to put in the search box of a Web search engine. Further complicating matters, aging—and age-related problems—don't occur in any predictable or uniform way. Oftentimes, it is difficult to ascertain *who* needs assistance and *when* they need it.

"The major problem arising from the aging process is that most sensory, motor, and cognitive abilities decline gradually with age, and at different relative rates for different individuals," observes Peter Gregor, professor of interactive systems design and dean of the school of computing at the University of Dundee in Scotland. What's more, "compared with younger adults, there is a wider diversity of characteristics among older people," says Gregor. This makes it more difficult to design a system to address specific issues. Factor in that older people didn't grow up with computers, smartphones, and other devices, and "the odds are stacked against them," Gregor concludes.

Designs on Usability

Addressing the computing challenges of older individuals requires ongoing analysis and creativity. Researchers know they must find ways to lighten the load on sensory and motor capabilities. Says Gregor, "We should be asking whether systems do all that is possible to minimize the cognitive load required to carry out tasks? Is support available if it is apparent a user doesn't know what to do next? Does the system support error-free learning? Do people feel that *they*, not the machine, are in

"Ultimately," says Peter Gregor, "operating a Web browser should be as straightforward as turning up the volume on a radio."

control of the interaction at all times?"

No less important, Gregor says, is to understand that older individuals are less enamored with the coolness of technology than they are about getting a specific task done. They also tend to treat machines with respect and are thus less likely to try things out—for fear of damaging or breaking something. "They are more prone to blame themselves when things inevitably go wrong." As a result, part of the focus for designers, engineers, and others is to educate and train older individuals to use systems effectively.

Wading through the tangle of issues is daunting, to be sure. A tool or feature that simplifies computing for one person may wreak havoc for another. For instance, using a larger font may create a longer page that involves more scrolling. A text-to-speech feature may eliminate the need to actually read the page but also test an older person's cognitive ability to comprehend everything he's hearing. What's more, if the system reads too fast or a person needs to replay a portion of the text and finds that he has to listen to the entire screen again (rather than being able to restart at a given point), he may give up.

However, some designers are beginning to take notice and develop viable solutions. For instance, IBM has introduced Easy Web Browsing (EWB), a set of features that make it simpler for older individuals to traverse the Internet in a user-friendly way. The browsing tool—used by more than 140 Web sites—serves as a bridge between standard Web site design and a format that takes into account factors such as vision loss and lack of experi-

Survey

Scientists and the Public

A survey of American scientists and the general public, conducted by the Pew Research Center for the People & the Press, has turned up some surprising results. The public rates scientists very highly, with 70% saying they contribute a lot to society's well being; only members of the military (84%) and teachers (77%) received a higher rating. However, scientists do not have such a high view of the public's scientific knowledge and expectations, with 85% of scientists viewing the public's lack of scientific knowledge as a major problem for science and nearly half of scientists blaming the public for unrealistic expectations about the speed of scientific achievements.

Many scientists fault the media for its science reporting, with 76% of scientists saying that a major problem for science is the media's failure to distinguish between results that are well founded and those that are not.

In terms of obstacles to high-quality research, 87% of scientists rate a lack of funding as an impediment to research. Moreover, 56% of scientists say that visa and immigration problems for foreign scientists and students hinder high-quality research. Both scientists and the public are in agreement on the importance of government funding of research, with 84% of scientists citing a government entity as an important source of funding for their research (49% cited the National Institutes of Health and 47% cited the National Science Foundation), while 60% of the public says government investment in research is essential for scientific progress. A majority of the general public believe that government funding of basic research (73%) and engineering and technology (74%) pay off in the long run.

The Pew Research survey was conducted via phone interviews with 3,006 members of the public and via an online survey of a random sample of 2,533 members of the American Association for the Advancement of Science, the nation's largest general scientific society.

ence. “It provides a more comfortable way for seniors to use the Internet,” Saito explains.

EWB, which has captured several industry awards, offers a number of advantages. It is easy to install, requiring the user to do nothing more than click a link. It’s easy to use, reading text aloud automatically when users point the mouse to an area of the Web page they want to read. It also presents controls in a convenient and consistent location on the screen for easy access, and provides a full screen mode that prevents the browsing screen from becoming hidden. Finally, it offers a high level of customization, including text magnification and the ability to read text aloud at different speeds and volumes.

Meanwhile, both Microsoft and Apple have built magnification tools, text-to-speech conversion, alternative keyboards, and specialized display options into their operating systems. Although these features are generally intended for those with disabilities, they’re also useful to many older individuals. In fact, for some, accessibility is what makes computer use possible in the first place. A study conducted by Microsoft found that one in four adults in the U.S. suffers from vision difficulties, one in four faces challenges with dexterity, and one in five has hearing problems.

Overall, Microsoft has developed more than 300 specialty assistive technology products for Windows computers. Not surprisingly, some of these tools provide sophisticated functionality. For example, reading tools now include software and hardware that,

among other things, can automatically reformat text so that it’s more easily viewed. Keyboard filters offer word prediction utilities to reduce typing and interaction. And light signaler alerts monitor sounds along with other events and alert users with a light signal. This makes it possible for a person with hearing problems to know when an email message has arrived or a computing task is completed.

The World Wide Web Consortium has also entered the picture. Its WAI-AGE project is currently studying Web accessibility barriers for older people. “Ultimately,” Gregor says, “operating a Web browser should be as straightforward as turning up the volume on a radio.”

Others are exploring ways to make input easier. IBM has developed keyboard software that monitors how a person is typing. Based on accuracy, speed, and other overall input patterns, it can adjust settings in a computer’s control panel. “If you are hitting keys over and over again, it learns to filter out the repeated keys for you,” Hanson explains. In addition, R&D continues on speech recognition, which could solve many of the interface problems plaguing the elderly. Saito at IBM is taking the concept a step further by studying speech symbolization, which creates icons or representations that serve as a bridge between human and machine interaction.

Not surprisingly, mobile phone manufacturers are also introducing devices that offer bigger and more prominent keys as well as larger text. Some, like the Samsung Jitterbug, also provides a simple “Yes” and

“No” menu system that reduces the complexity of the device—along with the cognitive demands placed on an older person. And a few organizations have worked to make their Web sites easier to navigate. For example, the National Institutes for Health’s NIH SeniorHealth site offers built-in tools for adjusting text size and contrast. It also provides a text-to-speech tool that reads pages aloud.

Hanson says the widespread belief that the problem will simply “go away” as the current generation of younger adults ages is entirely misguided. “Today’s older adults were proficient with the technology of their generation,” she says. “Technology is changing more rapidly now than it has at any time in the past. There is no reason to expect that future generations of older adults will be any better equipped to deal with new technology than today’s older adults are with today’s new technology.”

One thing is certain: addressing the needs and requirements of older individuals is paramount as employers, retailers, government, and others head online. Ultimately, it’s vital to recognize that gray matters and age counts. Concludes Czaja: “Older adults must be connected to society and we must ensure that they have access to information and opportunities. Researchers, designers, and engineers must find ways to make online information and services available to older adults.”

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

© 2009 ACM 0001-0782/09/0900 \$10.00

Networking

CCC’s Research Agenda for a Better Internet

The Computing Community Consortium’s Network Science and Engineering (NetSE) Council, led by Ellen W. Zegura, chair of computer science at Georgia Institute of Technology, has released *Network Science and Engineering (NetSE) Research Agenda*, a comprehensive report about the development of better networks, in particular the Internet, with the goal of

increased security, accessibility, predictability, and reliability.

“Literally hundreds of researchers contributed to the agenda by participating in workshops, authoring sections, and reviewing the overall document,” Zegura said via email. “While probably no one endorses every word, this deep engagement speaks to the entire research community’s appreciation of

the importance of ratcheting up networking research, and better supporting experimental efforts, long-term foundational efforts, and interdisciplinary efforts.”

The 116-page report is a living document, and the NetSE Council welcomes feedback and comments at <http://www.cra.org/ccc/netse.php>.

Zegura sees two primary research challenges. “First,

experimental research, and the tools and facilities required for that research, are not traditionally very well supported in our field,” Zegura said. “Second, in all fields—and ours is no exception—interdisciplinary research is hard to carry out. There are many areas of networking where only an interdisciplinary approach can make a significant dent in the problem.”

Computer Science Meets Environmental Science

Scientists share knowledge and seek collaborators at computational sustainability conference.

TWO HUNDRED ENVIRONMENTAL and computer scientists convened for four days in June for the First International Conference on Computational Sustainability, held at Cornell University. The conference's goal was to establish and develop a research community around the field of computational sustainability, which aims to develop computational and mathematical models and methods for the management of resources needed to solve the problems confronting sustainability in today's rapidly developing world.

As some conservationists and environmental scientists gave their presentations, however, it became apparent that their knowledge of the computational techniques applicable to the problems they want to solve lags behind the state of the art in computer science. Likewise, some computer scientists and mathematicians are unaware that ecological problems often translate into interesting decision optimization and statistical learning problems involving combinatorial decisions, dynamic modeling, and uncertainty, says Carla Gomes, director of Cornell's Institute for Computational Sustainability. "We must first find a common language," Gomes said. "This is new intellectual territory with great potential, and with unique societal benefits."

Several computer scientists who have created algorithms for environmental applications presented at the conference. Carlos Guestrin, a professor of computer science at Carnegie Mellon University and his former graduate student Andreas Krause (now an assistant professor of computer science at Caltech), for example, are optimizing the placement of sensors to detect contamination in drinking water distribution systems. They have also developed an algorithm that enables lake-trolling,




Carla Gomes, director of Cornell's Institute for Computational Sustainability, with associate director David Shmoys.

sensor-equipped robots to detect algal bloom and predict, even if no previous data exists, where it will occur next.

Vipin Kumar, head of the computer science and engineering department at the University of Minnesota, spoke about global scale patterns in biosphere processes and their impact on the global carbon cycle. He and colleagues at NASA are investigating the use of data mining algorithms to detect changes in the global land cover using satellite data. Kumar's team developed a novel recursive merging algorithm to identify changes in time series data, which they applied to the MODIS enhanced vegetation index for California from 2001 to 2008 and produced detailed information on forest fires, the conversion of farmland to residential areas, and the conversion of desert to farmland and other commercial uses.

Throughout the conference, environmental scientists encouraged computer scientists to collaborate with them. Michael Runge, a research ecologist at the U.S. Geological Survey's Patuxent Wild-

life Research Center, said he and his colleagues had believed there were no solutions to many of the complex ways they wanted to formulate ecological decision problems. "I've realized that we were over-constraining how we were thinking about problems," he said. "I've had my eyes opened to the number of tools available from the mathematics and computational side. The question is: How do we connect these amazing tools and the huge demand for their application to ecological problems?"

"We need people to bridge communication between all these fields, people who can see that a disease dynamics or water supply contamination problem looks a lot like a telecommunications network problem," says Runge. "We also need people to do the 'plug and chug' applied work that is not necessarily novel from the academic standpoint, but critical from the applied standpoint." 

Based in Manhattan, **Karen A. Frenkel** is a freelance writer and editor specializing in science and technology.

© 2009 ACM 0001-0782/09/0900 \$10.00

Law and Technology

Keeping Track of Telecommunications Surveillance

The creation of a statistical index of U.S. telecommunications surveillance activities and their results will benefit both civil liberties and law enforcement.

TELECOMMUNICATIONS SURVEILLANCE RAISES complex policy and political issues. It is also a matter of great concern for the general public. Surprisingly enough, however, the phenomenon of telecommunications surveillance is poorly measured in the U.S. at present. As a result, any attempt at rational inquiry about telecommunications surveillance is hampered by the haphazard and incomplete information the U.S. government collects about its own behavior and activities.

Neither the U.S. government nor outside experts know basic facts about the level of surveillance practices. As a consequence, U.S. citizens have limited ability to decide if there is too much or too little telecommunications surveillance. It is also impossible to determine if telecommunications surveillance is increasing or decreasing, or if law enforcement is using its surveillance capacities most effectively.⁴

Ideally, it would be possible to reach conclusions about these issues by examining data about U.S. govern-

ment surveillance practices and their results. As a general model, federal and state crime statistics are publicly available and criminologists pore over these databases to spot trends and determine police activities that are effective. No such database is available about the full range of telecommunications surveillance.

Congress should create one annual report card that measures and publicizes government's performance of telecommunications surveillance.

The Telecommunications Surveillance Index

Congress should create one annual report card that measures and publicizes government's performance of telecommunications surveillance. This index will replace the bits and pieces of scattered reports that different governmental entities sometimes release. Such an index will allow year-by-year comparisons of changes in the levels of government telecommunications surveillance and permit meaningful judgments about the extent of privacy invasions and the effectiveness of the activity. In this column, I describe the gap left by the reporting provisions in current statutes, which create only an incomplete and discontinuous picture of the governmental activity. The creation of an annual telecommunications surveillance index is an urgent matter, and I will conclude by discussing four issues related to this necessary task.

To understand the shortcomings of the statutes that permit U.S. telecommunications surveillance, one needs a

sense of how they collect information about government use. The critical statutory regulations are the Wiretap Act; the Pen Register Act; the Stored Communications Act; the Foreign Intelligence Surveillance Act (FISA); and the different provisions for National Security Letters. The first three laws concern the use of surveillance for domestic purposes—that is, in the context of ordinary criminal investigations. The last two statutes regulate the use of surveillance for foreign intelligence purposes, such as counterterrorism. And, in a nutshell, the most public information is generated about the U.S. government’s use of the Wiretap Act. Yet, this law in many ways has become less important than other telecommunications surveillance statutes, and we know far less about the use of these other statutes.

Telecommunications Surveillance for Criminal Investigations

A review of the legal basis for telecommunications surveillance starts, logically, with the Wiretap Act, which is the oldest of the modern statutory authorities in this area. Enacted in 1968, the Wiretap Act sets a high statutory standard before the government can “intercept” a “wire or oral communication.” It also requires the government to publish relatively detailed data sets about its use. The Wiretap Act assigns the task of collecting this information to the Administrative Office of the United States Courts, which then publishes the statistics.¹

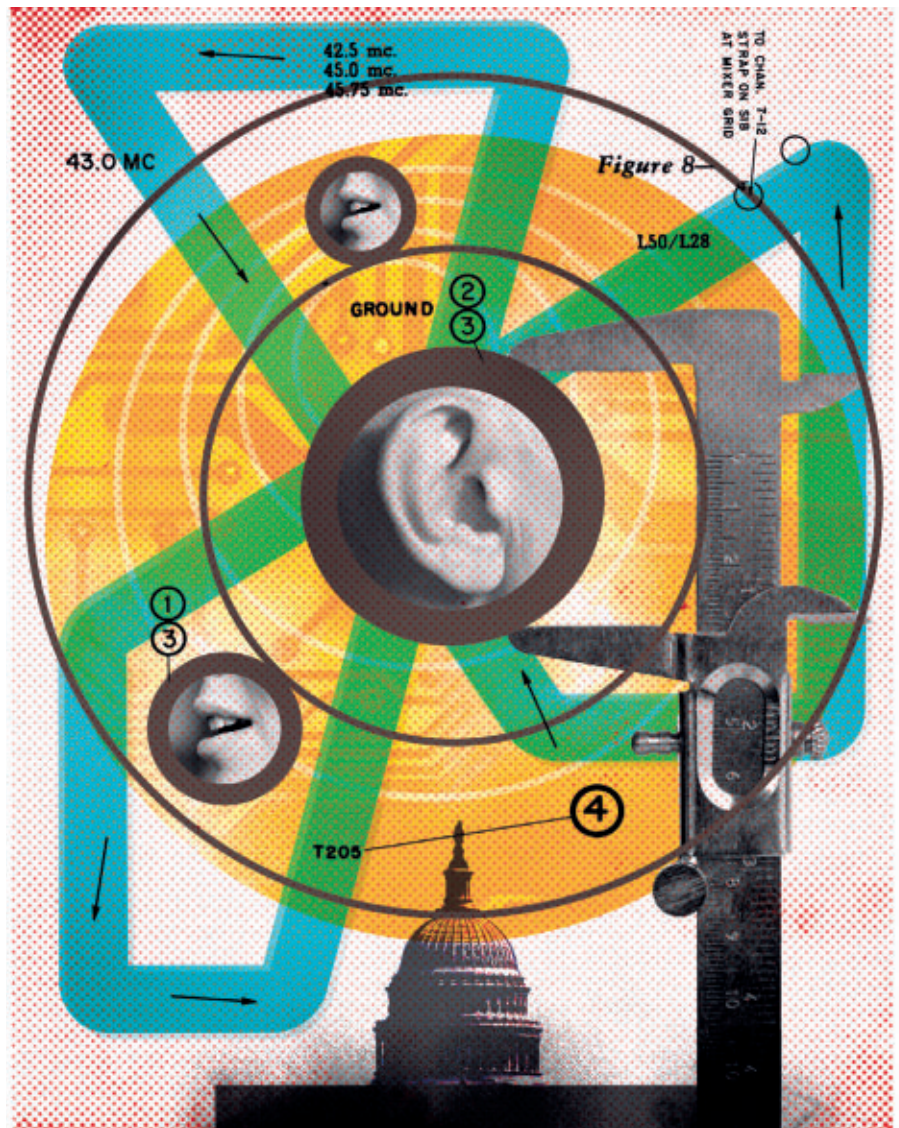
What is the problem then? The difficulty is that the Wiretap Act regulates only the capturing of the content of messages contemporaneously with their transmission. As an example of its coverage, if law enforcement wishes to intercept a telephone call as it is occurring, the Wiretap Act will apply. Yet, technological changes have created a variety of information that falls outside the Wiretap Act, whether because it is “telecommunications attributes” rather than content, or stored on a server. Telecommunications attributes are generally regulated by the Pen Register Act, and information stored on a server generally falls under the Stored Communications Act. I will consider each law in turn.

The Pen Register Act, as first en-

acted in 1986, regulated only access to telephone numbers dialed from a specific phone, or received by it. Today, the Pen Register Act, as amended by the Patriot Act in 2001, more broadly regulates access to “dialing, routing, addressing, or signaling information.” Examples of such information are IP addresses and email addressing information.

all, the situation is reminiscent of the anarchic administrative conditions prior to the New Deal’s creation of the Federal Register and other means for the orderly publication of governmental records.

As a further shortcoming, pen register reports only list *federal* collection of information pursuant to the law. If use of the Pen Register Act follows the



Like the Wiretap Act, the Pen Register Act requires collection of information about its use. Yet, reports pursuant to it are far less detailed than those under the Wiretap Act, and the U.S. government does not make them publicly available. And perhaps the greatest surprise is that Congress has shown scant interest in even ensuring it actually receives the information to which it is statutorily entitled from the Department of Justice. Over-

pattern of the Wiretap Act, however, states are now engaging in far greater use of their authority than are federal law enforcement authorities.

The third statutory authority for telecommunications surveillance is the Stored Communications Act. This statute is particularly significant today because so many kinds of telecommunications occur in asynchronous fashion. For example, sending an email message may be the most prevalent

form of telecommunications in the U.S. today. Yet, an email message is in transmission, as the term is understood under the Wiretap Act, for only a short period. Transmission is the time it takes from clicking on the “send” command to the moment the message arrives at the server of the recipient’s ISP. Rather than recourse to the Wiretap Act, law enforcement typically seeks collection of email from ISPs under the Stored Communications Act, which contains requirements for obtaining access to information that are generally less rigorous than under the Wiretap Act.

Despite the centrality of the Stored Communications Act, there are almost no official statistics collected about law enforcement’s use of this statute. This statute contains only a single reporting exception, which regards disclosure in an emergency. Information about its use is given to House and Senate committees, but is not made publicly available at present. In this regard, Switzerland offers a step in the right direction: in that country, the Federal Department of Justice and the police publish annual information about the number of orders for stored information.²

Telecommunications Surveillance for Foreign Intelligence Purposes

The three statutory authorities thus far surveyed all regulate access to telecommunications information for domestic law enforcement purposes. On the intelligence side, FISA provides the chief statutory regulation for the government’s collection of information about foreign intelligence within the U.S. In addition to FISA, several stat-

The annual index should include information about all statutory authorities, not just the Wiretap Act.

utes permit the FBI to obtain personal information from third parties through National Security Letters (NSLs). A NSL is a written directive from the FBI in cases involving national security; it does not require judicial review.

FISA requires the Department of Justice to file annual reports with Congress and the Administrative Office of the Courts. These reports provide merely skeletal information about the use of FISA authorities. FISA also requires the Attorney General to file reports with the Senate and House regarding all uses of pen register devices, pursuant to this statute. This information is made publicly available.

As for the NSLs, in its reauthorization of the Patriot Act in 2005, Congress required two important kinds of information to be collected about this kind of information gathering. First, it expanded an existing reporting requirement that sent information to Congress, and required annual public data on the FBI’s request for NSLs. Second, the law required the Department of Justice to carry out audits of the use of NSLs. The resulting audits have already demonstrated substantial underreporting of the actual number of NSLs and misuse of statutory authorities.

Steps to Take

As I’ve described here, there is currently inadequate data about telecommunications surveillance in the U.S. I conclude by discussing four themes related to creation of a national telecommunications surveillance index. First, a central role should be given to the Administrative Office of the U.S. Courts, as under the Wiretap Act, in collecting and publicizing telecommunications surveillance statistics. Since 1968, the Administrative Office has successfully carried out this role pursuant to the Wiretap Act, and the other applicable statutes should be amended so that applicable information goes to this entity.

Second, the annual index should include information about *all* statutory authorities, not just the Wiretap Act. As noted earlier, this statute is less important as a source of statutory authorization for surveillance activity than the Stored Communications Act and other statutes.

Third, one of the most difficult tasks in creating an annual report card will be harmonizing the information collected within a single index. The goal is clear: to provide a picture of how activities in different statutory areas relate to each other. Nonetheless, development of a workable yardstick raises a series of complex issues because each statute sweeps in different kinds of data and, sometimes subtly, different kinds of surveillance.

Fourth, telecommunications surveillance statutes should increase independent audit functions. It is essential to have an independent assessment of the accuracy of the supplied data and the completeness of supplied reports. As part of this assessment, the use of statistical sampling of case files will be a useful technique. The Inspector General of the Department of Justice has already taken this approach in assessing use of NSLs pursuant to its audit authority. In an international illustration of this methodology, the Max Planck Institute for Foreign and International Criminal Law published an ambitious statistical analysis of a sample of telecommunications surveillance orders issued in Germany.³

The twin goals of an annual telecommunications surveillance index should be to minimize the impact of surveillance on civil liberties and to maximize its effectiveness for law enforcement. There is a compelling need at present for Congress to require statistical benchmarks to accompany all the laws that authorize telecommunications surveillance. ■

References

1. Administrative Office of the United States Courts. *2007 Wiretap Report*; <http://www.uscourts.gov/wiretap07/contents.html>.
2. Eidgenössisches Justiz und Polizeidepartement, Überwachung des Post und Fernmeldeverkehrs; http://www.ejpd.admin.ch/ejpd/de/home/themen/sicherheit/ueberwachung_des_post/statistik.html.
3. Albrecht, H.J., Grafe, A., and Kilching, M. Max-Planck-Institut für ausländisches und internationales Strafrecht, Rechtswirklichkeit der Auskunfterteilung über Telekommunikationsverbindungsdaten nach §§ 100g, 100h StPO (March 2008), Deutscher Bundestag, Drucksache 16/8434.
4. Schwartz, P.M. Reviving Telecommunications Surveillance Law. *University of Chicago Law Review* 287 (2008); <http://www.paulschwartz.net/pdf/12%20Schwartz%20Final%202.19.pdf>.

Paul M. Schwartz is a professor of Law at the University of California, Berkeley Law School, and a director at the Berkeley Center for Law and Technology.

Copyright held by author.

The Profession of IT Computing: The Fourth Great Domain of Science

Computing is as fundamental as the physical, life, and social sciences.

THERE IS MUCH discussion about the generality and pervasiveness of computing. Is computing really an inescapable part of the world? What does that imply about science? Engineering? Education? How can we build new stories and education experiences that attract new young people to the field? Can computing, like other sciences, advance technology and applications through strong scientific advances? Can computer science rightfully claim a place at the table of science? And so on.

Many people have been warming up to the ideas that computing is science,

deserves a place at the table of science, and is a rewarding profession. Yet a question nags at the edge of perception. Computers are admittedly everywhere. Roads, electricity, radio, television, and food are everywhere too, but they are not science. They are infrastructure. Why is computing any different?

We recently discovered a new answer to this old question. We noticed that all the acknowledged sciences are grouped into three great domains: physical, life, and social. We asked, what makes them great domains of science? And we found that computing meets all the same criteria. In other words, computing is the

fourth great domain of science.

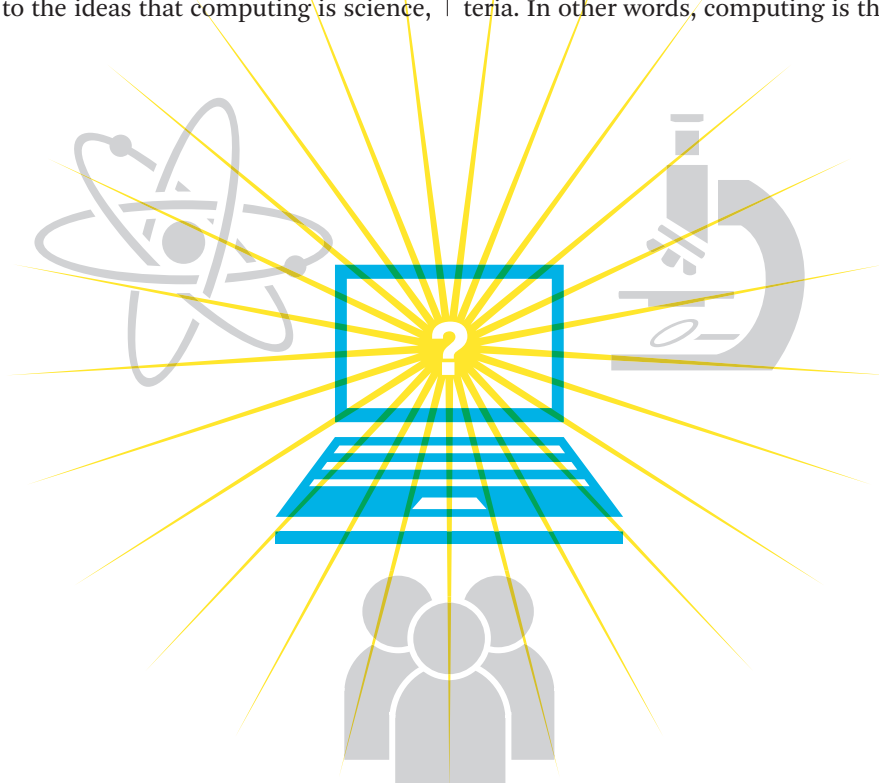
We will show you why we make this claim. We hope that you will not only want to discuss it, but that you will warm up to it too.

Great Scientific Domains

Most of us understand science as the quest to understand what is so about the world. Through observation and experimentation, scientists seek to discover recurrent phenomena. They formulate models to capture recurrences and enable predictions, and they seek to validate or refute models through experiments. Much of computing conforms to these ideals.^{7,10}

Science has a long-standing tradition of grouping fields into three categories: the physical, life, and social sciences. The physical sciences focus on physical phenomena, especially materials, energy, electromagnetism, gravity, motion, and quantum effects. The life sciences focus on living things, especially species, metabolism, reproduction, and evolution. The social sciences focus on human behavior, mind, economic, and social interactions.⁸ We use the term “great domains of science” for these categories.⁹

These domains share three common features: their foci are distinctive phenomena important in all sciences; the fields of each category have rich sets of structures and processes that evolve together through constant interaction; and their influence is extensive, touching all parts of life and providing unique and useful perspectives.



Examples of computing interacting with other domains.⁸

	Physical	Social	Life	Computing
Implemented by	mechanical, optical, electronic, quantum, and chemical computing	Wizard of Oz, mechanical turks, human cognition	genomic, neural, immunological, DNA transcription, evolutionary computing	compilers, OS, emulation, reflection, abstractions, procedures, architectures, languages
Implements	modeling, simulation, databases, data systems, digital physics, quantum cryptography	artificial intelligence, cognitive modeling, virtual humans, autonomic systems	artificial life, biomimetics, systems biology	
Influenced by	sensors, scanners, computer vision, optical character recognition, localization	mouse, keyboard, learning, programming, user modeling, authorization, speech understanding	eye, gesture, expression, and movement tracking, biosensors	networking, security, parallel computing, distributed systems, grids
Influences	locomotion, fabrication, manipulation, open-loop control	screens, printers, graphics, speech generation, network science, cognitive augmentation	bioeffectors, haptics, sensory immersion	
Bidirectional Influence	robots, closed-loop control	human-computer interaction, full immersion, games	brain-computer interfaces	

Computing does not seem to fit nicely into any of the traditional domains. Computation is realized in physical media and is even part of some physical processes (for example, quantum mechanical waves). More recently computation has been found in living systems (for example, DNA transcription) and social systems (for example, evolution of scale-free networks). Although computational methods are used extensively in all the domains, none studies computation per se—computation is not a physical effect, a living entity, or a social entity.

What if computing is a separate domain? It satisfies the three criteria. It

Computing interacts not only with people and other living systems, but with the physical world.

has a distinctive focus—computation and information processes. Its constituent fields—computer science, informatics, information technology, computer engineering, software engineering, and information systems—and its structures and processes are in constant interaction. Its influence is pervasive, reaching deep into people's lives and work.

The core phenomena of the computing sciences domain—computation, communication, coordination, recollection, automation, evaluation, and design^{2,6}—apply universally, whether in the artificial information processes generated by computers or in the natural information processes found in the other domains. Thus, information processes in quantum physics, materials science, chemistry, biology, genetics, business, organizations, economics, psychology, and mind are all subject to the same space and time limitations predicted by universal Turing machines. That fact underpins many of the interactions between computing and the other fields and underlies the recent claim that computing is a science of both the natural and the artificial.³⁻⁵

It might be asked whether mathematics is a great domain of science. Although mathematics is clearly a great domain, it has traditionally not been considered a science.

The Nature of Interactions

Two out of the three criteria listed earlier involve interactions, either among structures and processes or among domains. These interactions generate the essential richness of science. They also complicate how we observe and understand science.

Hierarchical taxonomies are the usual ways of observing a domain of science. It is easy to craft a tree hierarchy representing all the parent and child relationships among fields in the domain. For example, the physical sciences are partitioned into chemistry, physics, astronomy, geology, etc., and each of those may be partitioned further, for example, regular and organic chemistry. Each field has its own “body of knowledge,” often represented with a taxonomy or a tree. Computing is likewise divided into constituent fields and subfields, each with a body of knowledge.

Hierarchical structures are very good for understanding static aspects of a field, but not its dynamics. Within a field, the interesting phenomena are not simply the properties of “things”; they are interactions among multiple things. Chemistry is not simply chemicals; it is the reactions among elements. Mechanics is not simply gears and levers; it is the forces among these parts. Psychology is not simply emotions, urges, and mental states; it is transactions and relationships. Similarly, computing is not just algorithms and data structures; it is transformations of representations.

The interactions are the real action of a field. Their complexities and uncertainties demand constant experimentation and validation in science and engineering. They make things messy and unpredictable. They are sources of innovation.

Scientific phenomena can affect each other in one of two ways: *implementation* and *interaction*. A combination of existing things implements a phenomenon by generating the intended behaviors of the phenomenon. Digital hardware physically implements computation.

Artificial intelligence implements aspects of human thought. A compiler implements a high-level language with machine code. In chemistry, hydrogen and oxygen implement water. In molecular biology, complex combinations of amino acids implement life.

Interaction occurs when two phenomena influence each other. In physics, atoms arise from interactions among the forces generated by protons, neutrons, and electrons. In astronomy, galaxies interact via gravitational waves. In computing, humans interact with computers.

Interactions exist not only within domains but across domains. Computing is implemented not only by physical processes, but by life processes (for example, DNA computing) and social processes (for example, games that produce outputs¹). Likewise, computing can implement, or at least simulate, structures and processes in these other domains. Computing interacts not only with people and other living systems, but with the physical world (for example, through sensor networks and robots).

The accompanying table illustrates a wide range of implementation and interaction relationships between computing and the four domains (including itself). An entry in the table identifies a way that computing interacts with a domain. For example, the entry for “quantum computing” in the “Physical” column and “implemented by” row means that computation is implemented by quantum processes from the physical sciences domain.

The examples in the table are sufficient to demonstrate the amazing extent of interactions between computing and the other domains. Computing is much more than infrastructure; it is an equal partner that strongly influences thought, practice, and approach.

There is still more to the story. Many other interactions involve more than two fields or domains. For example, the emerging field of “network science” is built on multi-way interactions among the computing, physical, and social sciences.

Down to the Basics

We’ve used the term “computation” as if everyone agrees on its meaning. In fact, this is not so. Typical definitions include “activity of a computer,” “phe-

To say that computing is a domain of science does not conflict with computing’s status as a field of engineering or even mathematics.

nomena surrounding computers,” and “transformation of content.” None of these captures all the notions of computing we can see at work in the table. For example, biologists believe DNA encodes information about the living body and that DNA transcription is a natural information-transforming process that creates the amino acids that generate new life. They clearly do not think of information as something stored in a computer database or transcription as an activity of a computer.

One way to define computing in a sense broad enough to cover everything in the table is to base it on the evolution of representations.⁶ Except in artificial intelligence, representations are a somewhat neglected aspect of computing. Computing scientists need to get better answers to key questions. What do we mean by a representation? What does it mean to represent something in a computationally amenable format? Should representations be grounded in the world, or projected from a mathematical definition? What is not a representation? In what way is computation an evolution of representations?

In fact, representations are not the only fundamental principle of computing in need of new answers. Many of the oldest questions are being reopened.¹¹ What is computation? What is information? What is intelligence? How can we build complex systems simply?

Conclusion

Computing is pervasive because it is a fundamental way of approaching the world that helps understand its own cru-

cial questions while also assisting other domains advance their understandings of the world. Understanding computing as a great domain of science will help to achieve better explanations of computing, increase the attraction of the field to newcomers, and demonstrate parity with other fields of science.

To say that computing is a domain of science does not conflict with computing’s status as a field of engineering or even mathematics. Computing has large slices that qualify as science, engineering, and mathematics. No one of those slices tells the whole story of the field.

The exercise of examining computing as a domain of science reveals that the extent of computing’s reach and influence cannot be seen without a map that explicitly displays the modes of implementation and interaction. It also reveals that we need to revisit deep questions in computing because our standard answers, developed for computer scientists, do not apply to other fields of science. Finally, it confirms that computing principles are distinct from the principles of the other domains. ■

References

1. von Ahn, L. Games with a purpose. *IEEE Computer Magazine* (June 2006), 96–98.
2. Denning, P. Great principles of computing. *Commun. ACM* 46, 11 (Nov. 2003), 15–20.
3. Denning, P. Is computer science science? *Commun. ACM* 48, 4 (Apr. 2005), 27–31.
4. Denning, P. Computing is a natural science. *Commun. ACM* 50, 7 (July 2007), 13–18.
5. Denning, P. Beyond computational thinking. *Commun. ACM* 52, 6 (June 2009), 28–30.
6. Denning, P. and Martell, C. Great Principles of Computing Project; <http://cs.gmu.edu/cne/pjd/GP>
7. Newell, A., Perlis, A.J., and Simon, H.A. “Computer Science,” letter in *Science* 157, 3795 (Sept. 1967), 1373–1374.
8. Rosenbloom, P.S. A new framework for computer science and engineering. *IEEE Computer* (Nov. 2004), 31–36.
9. Rosenbloom, P.S. *Computing as a Great Scientific Domain: A Multidisciplinary Perspective*. To be published (2009).
10. Simon, H.A. *Sciences of the Artificial, Third Edition*. MIT Press, Boston, MA, 1996.
11. Wing, J. Five deep questions in computing. *Commun. ACM* 51, 1 (Jan. 2008), 58–60.

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

Paul S. Rosenbloom (rosenbloom@usc.edu) is a professor of computer science in the Viterbi School of Engineering at the University of Southern California, a project leader at the USC Institute for Creative Technologies, and the former deputy director of the USC Information Sciences Institute.

Copyright held by author.

Emerging Markets

How ICT Advances Might Help Developing Nations

Some predictions for technology developments, deployments, and the associated societal implications.

PEOPLE MAKE PREDICTIONS in hope (and sometimes despair) or in expectation. These simple speculations are about technology, about how the art of the possible has a habit of becoming the everyday, and about how certain advances might help the developing world. But they are toward the hope end, rather than the expectation end, of that spectrum—because technology alone is not enough. To take optimum advantage of these and other advances we need to think holistically and must develop some new business models. I encourage you to think creatively about not just what capabilities the information communication technologies (ICT) developments I discuss in this column might enable but the new ways we might organize to deliver them.

Connection will become more pervasive. The world is connecting. Slowly, in many places, but every day we understand better how to bring more parts of the world into the connected space. So look ahead and hope—perhaps expect—that nations currently underserved will be able to join the networked world, across networks that are fast, always on, and pervasively available through low-cost devices, accessing systems that are more intelligent, easier to use, with more natural user interfaces.

We need this connecting more than ever because we face difficult challenges—differently felt in different nations



at different times, but nonetheless intimidating in their scope and impact. Among them are six major forces presenting governments of the world with their toughest problems in the next decade or so: accelerating globalization, changing demographics, rising environmental concerns, evolving societal relations and expectations, growing threats to social stability and order, and the impact of new technology. And now the world also faces economic

disruption in the form of broad and deep recession that promises to shrink global gross domestic product.

Cloud computing will help developing nations “leapfrog” the developed world. As connections become more available, cloud computing can become an important platform through which emerging nations can access modern government capabilities and systems. All governments need financial management and administration as well as

business and citizen services such as permitting, licensing, benefits determination and distribution, and health information. But those without them might not need to acquire them the way the developed world has—somewhat haphazardly over the years, with large capital and operational spending on data centers and heavily customized suites of application software, with legacies that do not easily link.

Not today, and not quite tomorrow—but when we have worked out the issues around what software can really take advantage of the cloud model, and what government concerns there are around security, location of data, and so on—will the cloud become a cost-effective platform and a way to deliver significant changes in governments' capabilities without huge cost? Tying a cloud platform together with mobile devices creates a versatile and cost-effective platform for all kinds of services.

Perhaps cloud centers, regionally placed, with public and private investment, could allow the developing world to access the kinds of systems that help developed nations' governments—but without the “legacy” and level of investment that it took. Perhaps they could show international organizations some ways to better leverage their investments. I hope such approaches could be game-changing in how we help nations improve transparency and accountability, enhance citizen services, and generate economic development.

Early examples are proliferating—in Vietnam a government agency uses a cloud approach to provide a collaborative infrastructure for innovation: linking government, universities, private-sector research, startups, and other organizations. And a Chinese city wanting to create a software industry for the region plans to support several hundred thousand developers across hundreds of companies—but it won't build a data center to do it. It will provide a cloud-based development environment that can scale as the vision grows.

“Mashing up” medical services, consumer electronics, and connectivity will allow broader and more cost-effective access to health care. Telepresence is already enabling remote consultations. Over time, new, smaller, less-expensive sensors will allow condition monitoring

We are learning how ICT can be applied in new ways to diverse environmental challenges.

of potentially millions of patients. We'll see new ways of navigating medical information drawn from multiple sources, and in particular radical new visualizations—think of a “Google Earth” for the body. Already researchers have demonstrated prototype visualization software that allows doctors to interact with medical data the same way they interact with their patients: by looking at the human body—through a 3D avatar.

In another example, to overcome the problems of deploying expensive imaging equipment in many different places, imaging and diagnosis will be able to use relatively inexpensive and accessible components, then use the network to access high-performance computing facilities to create key images and transmit them. Splitting information collection, processing, and visualization will enable much wider deployment into remote and less prosperous parts of the world.

The immersive Internet—becoming a multidimensional place. Early virtual worlds, including games, are precursors to a “3D” Net, one that integrates with the existing Web and allows for new applications with enhanced immediacy and interactivity. Such an environment will encourage the formation of in-world social groups—collaborations, teams, guilds, clubs, neighborhoods, and so forth. The Internet will go even further toward satisfying two key aspects of being human: our innately social and visual natures.

The developed nations are exploring the retail possibilities of the 3D Internet—an immersive world where you'll “walk” the aisles of supermarkets, bookstores, and other shops and you'll encounter experts you'd rarely find in your local store.

But beyond that, the 3D Internet will enable whole new kinds of interactive education, remote medicine and citizen access, transforming how people can interact with our friends, family, doctors, teachers, government, and more.

The initial hype has abated a little. But it is still early, and there remains much promise for the immersive, social attributes of these virtual worlds, which, in a connected world, can reflect real-life experiences and bring new levels of education and training to remote and underserved communities.

Technology will help to tackle environmental and resource challenges. We are learning how ICT can be applied in new ways to diverse environmental challenges. We know that the developing world, while not alone in this, is at the forefront of tackling energy, environment, and resource problems. So what might be possible?

More effective water filtration is emerging from nanotechnology research. As we think more about water management and conservation as an information problem, using sensors and actuators connected across networks to large-scale computing resources, it changes our approaches to managing quality and supply.

Intelligent transportation systems are proving they can reduce congestion and cut greenhouse gas emissions. Intelligent grids make better use of power, wasting less and allowing easier access to alternative generation sources.

Silicon technology research is finding new ways to build solar power technology. Have you ever considered how much energy could be created by having solar technology embedded in roads, in the frames of buildings, in paint, rooftops, and windows? Until now, the materials and the process of producing solar cells for solar energy conversion have been too costly for widespread adoption. But now this is changing with the creation of “thin-film” solar cells, a new type of cost-efficient solar cell that can be 100 times thinner than silicon-wafer cells and produced at a lower cost. These new thin-film solar cells can be “printed” and arranged on a flexible backing, suitable for not only the tops, but also the sides of buildings, tinted windows, cellphones, notebook computers, cars, and even clothing.

ACM Journal on Computing and Cultural Heritage



JOCCH publishes papers of significant and lasting value in all areas relating to the use of ICT in support of Cultural Heritage, seeking to combine the best of computing science with real attention to any aspect of the cultural heritage sector.



www.acm.org/jocch
www.acm.org/subscribe



Association for
Computing Machinery

Imagine being within a phone call's reach from the ability to post, scan, and respond to email and instant messages—without typing.

For the developing world—perhaps, one day, a village school might “bid” for cheaper electricity cycles—and the solar cells of a neighboring village might provide them.

People will talk to the Web...and it will talk back. People will be able to use the Internet through natural voice interaction—eliminating the need for visuals or keypads. New technology will change how people create, build, and interact with information and e-commerce Web sites—using speech instead of text. We know this can happen because the technology is available, but we also know it can happen because it must. In places like India, where the spoken word is more prominent than the written word in education, government, and culture, “talking” to the Web is leapfrogging all other interfaces, and the mobile phone is outpacing the PC.

Researchers creating the “spoken Web” will enable people without traditional access to the Internet, but with access to a mobile or landline phone, to gain access to a worldwide collection of “voice sites”: Web sites accessible by voice commands over a telephone network. It particularly helps people who are not able to read or write, the elderly, and the economically disadvantaged. It has enormous potential, for example, for providing ways that village communities can offer their products and services worldwide using a voice-enabled Web portal.

Imagine adding to the spoken Web advances in language translation, speech recognition, and speech synthesis. During the Beijing Olympics, visitors could use a modified mobile

device in a novel way: they could speak their destination and the machine would recognize, translate, and synthesize the Mandarin equivalent for their taxi driver. U.S. soldiers in Iraq have pilot-tested handheld translators to take spoken English and produce spoken Arabic, or vice versa.

And once the Web is more accessible by using voice, it will become easier to use for everyone. Imagine being within a phone call's reach from the ability to post, scan, and respond to email and instant messages—without typing. Think of being able to search the Web verbally and have the information read back to you, just as if it were an actual conversation.

Conclusion

These are speculations, not guarantees. And they are selections, taken from the broad universe of technology advances. But they might illuminate two important things I hope we can achieve. Of one I am pretty sure—the world will continue to become more interconnected, instrumented, and intelligent. Of the other: well, who needs to benefit the most from those attributes? If we have learned anything from recent trials, it is that the big problems of the world are intimately intertwined. Climate change knows no borders; shifting demographics affect us all, one way or another; and we all suffer when the poorest suffer. So my conjecture is that these elements of the increasingly connected world will enable those in the poorest nations to participate in ways they have not experienced before.

But to use those capabilities responsibly and effectively, we will need creativity—to conceive of the new solutions we can build for the developing world as much as for the developed; collaboration—to break down the boundaries that inhibit change, and to leverage the minds and skills of the many; and courage—to move beyond current business models to drive desirable change. □

Mark Cleverley (mark.cleverley@us.ibm.com) is a solutions executive with IBM Global Government Industry.

The writer's views are his own—though not exclusively—and do not necessarily represent the views of IBM Corporation.

Copyright held by author.



DOI:10.1145/1562164.1562178

Cameron Wilson and Peter Harsha

IT Policy

The Long Road to Computer Science Education Reform

Viewing the factors impeding improvements to CS education from kindergarten through grade 12 from a policy perspective.

WHENEVER COMPUTER SCIENTISTS discuss the human capital issues that plague the computing fields—the significant lack of diversity among computer science graduates (especially at the master’s and Ph.D. levels),^a the underrepresentation of women in computing, and the general decline in interest among U.S. students in the science and engineering fields—inevitably, the lamentable state of computer science education in the U.S.’s primary and secondary schools figures prominently.

While there are significant issues within higher education, there is a growing realization that we must address challenges at the beginning of the education pipeline. Excellent computer science classes are being taught in U.S. schools today, but looking across the country they are the exception to the rule. In general, we find too few students have the opportunity to take engaging and rigorous computer science courses in high school; there is little diversity among those that do. Too few opportunities exist for professional development for teachers. Too little innovation has happened in creating an engaging and rigorous curriculum for

a Traditionally underrepresented minorities (Black or African-American, Hispanic, American Indian or Alaska Native) are even more underrepresented at the master’s and Ph.D. levels, according to the 2007–2008 CRA Taulbee Survey; see <http://www.cra.org/taulbee/CRA-TaulbeeReport-StudentEnrollment-07-08.pdf>



Approximately 200 7th–10th grade students from around the U.S. attended the 2009 Summer Science, Technology, Engineering and Math (STEM) Program at the U.S. Naval Academy in Annapolis, MD. The five-day program offered real-life applications in subjects including forensics, mechanics, robotics, biometrics, and computer stimulation and encourages students to pursue engineering and technology studies in high school and college.

students. There is general agreement that this is a national failing—and one that we can ill afford—as computing is a central part of society, and key enabler of innovation and economic growth.

If “fixing” computer science education in kindergarten through grade 12 (K–12) is so clearly necessary, why has there not been more progress in the U.S.? In an age when the ability to think computationally already is, or certainly will be, a prerequisite for success in so many endeavors, why do we still strug-

gle to reform K–12 computer science and make it more relevant?

In large part, it is because reform of the K–12 education system at any level or in any subject is notoriously difficult. Control over education is decentralized. States and school districts play varying leadership roles in determining what students must learn. Federal policy and bureaucracy, driven largely through strings attached to federal funds, layer on top of state and local responsibilities. Add to this

mix numerous outside organizations from teachers' advocates to parent groups demanding a range of reforms and the result is an immensely complex web of policies, institutions, and players shaping the U.S. education system. From the outside looking in, any large-scale reform effort seems doomed for marginalization within this behemoth.

To further complicate the calculus, pressure is increasing, both nationally and internationally, for countries to take immediate steps to strengthen science and mathematics education to foster future innovation in high-technology fields.^b President Barack Obama made this national push plain in a major speech to the May 2009 annual meeting of the prestigious National Academies in Washington D.C. stating, "since we know that the progress and prosperity of future generations will depend on what we do now to educate the next generation, today I'm announcing a renewed commitment to education in mathematics and science."^c This statement moved discussions of improving science, technology, engineering and mathematics education (also known as STEM) to the forefront of national debates about education reform.

With the stage thus set for STEM education reform, two important questions come to mind: First, is this going to be the classic case of the irresistible

force (the need for coordinated STEM reform) meeting the immovable object (state and local control of education)? Second, and most relevant to the computing community, how does computing fit into overall K-12 STEM education reform? Before we can begin to answer these questions it is useful to understand the basic workings of the education system in the U.S.

Any reform discussion starts with two important state-based concepts: standards and assessments. These are the backbone of the education landscape. Each state sets learning standards for students in the state's K-12 school system. For example, one part of the state of Virginia's sixth-grade mathematics standard is that students should be able to "...identify, represent, order, and compare integers."^d Then, in most states, it is up to the school districts to establish curriculum implementing these standards. State and school districts assess whether the concepts are learned through testing. And yes, just because something is in the standards does not mean students will be exposed to it, and just because they are tested does not mean they know it. The point is to understand the education policy framework because it looms large in efforts to reform education.

Graduation requirements are equally important. Most states set or provide guidance on the credits students must accumulate to graduate from high school (also known as secondary school). These requirements fall into two general categories: a set of "core" courses that students must take to graduate; and electives that constitute everything not in the core. For example, the state of Texas requires that students have four years of high school mathematics credits in order to graduate. California aligns its graduation requirements with the higher education system by mandating a set of courses that are the minimum admission requirements to state-funded schools.

While states and/or local governments generally make graduation and curriculum decisions, these decisions are influenced by national goals and accountability requirements of certain federal laws. The most notable federal

law is the controversial No Child Left Behind Act. Its accountability provisions require that states test in reading, mathematics, and science at certain grade levels. If individual schools do not meet state-based benchmarks for student achievement in reading or mathematics then those schools can face federal sanctions.

Because standards, assessments, curriculum, and graduation requirements are state and/or local decisions, advocates for national STEM reform face a difficult challenge of making their case to policymakers in each state, or even to the 14,000 school boards across the nation. Local governance of education has rendered any discussions of "national standards" political nonstarters. However, as President Obama's speech signified, the political landscape for reform has begun to change. Two years ago, the National Science Board identified the crises in STEM education^e stating that the system was failing America's youth and curriculum needed reform and coordination both within and among the states. Soon after the president's speech, powerful state education groups announced that 46 states would work toward harmonizing (not nationalizing) some standards.^f The administration is also putting money on the line with \$5 billion targeted for overall education reform efforts, some of which will likely go to STEM education. All of this points to a solution that is driven from the bottom, but with top-down pressure and incentives being applied.

However, there are some important caveats to this progress—particularly for computer science education. First, harmonizing standards between states is politically difficult because standards affect what is assessed and therefore tested. Scores on those tests can affect federal funding and parents' perceptions of schools. Second, of the \$5 billion in new federal education funding, it is not clear how much of it will go to STEM education reform. Third, it appears that states will focus initially on harmonizing math and reading standards, then turn to reviewing science

^b This was one of the key conclusions of ACM's Globalization and Offshoring report: <http://www.acm.org/globalizationreport/>

^c April 27, 2009, remarks by President Obama, http://www.whitehouse.gov/the_press_office/Remarks-by-the-President-at-the-National-Academy-of-Sciences-Annual-Meeting/

Any reform discussion starts with two important state-based concepts: standards and assessments.

^d <http://www.doe.virginia.gov/VDOE/Superintendent/Sols/math6.pdf>

^e <http://www.nsf.gov/nsb/stem/>

^f <http://www.washingtonpost.com/wp-dyn/content/article/2009/05/31/AR2009053102339.html>

standards. Most states do not have a specific set of computer science standards, but even if they exist, the state's focus on the "common core" likely will not include existing computer science standards. We need to delve even deeper into the education system to understand this.

Standards, graduation credits, and No Child Left Behind drive students and administrators toward emphasizing—both in terms of what students take and resources dedicated to developing them—the "core" courses of the curriculum. It is a gross oversimplification of an incredibly complex system to say that students across the nation are taking a similar set of core courses. The key issue for computer science education is, as a general rule, computing is absent from the "core." Much of what is called computing education by states at the K–12 level, particularly high school, is placed within the technology curriculum both in the states standards and the schools. However, the curriculum of so-called computing classes within this category largely focuses on the use of technology (keyboarding, or learning word processing/spreadsheets) instead of core computing concepts. Further, technology classes are generally elective credits for students on par with health or shop class.

This categorization puts efforts to get rigorous computing courses into the college-bound academic curriculum at a significant disadvantage. What is considered technology in school is typically not an academic subject area for the college-bound student; rather classes to help bolster vocational education for those about to enter the work force. Students pursuing college often do not have the time for elective credits, particularly those focused on a vocation.

Despite these daunting obstacles, there are exciting efforts already under way led by different parts of the community to address this national failing:

- ▶ Two years ago, ACM created an Education Policy Committee^g to focus on public policy issues in science and math education relevant to computing and computer science. Since its inception, the committee has identi-

g <http://www.acm.org/public-policy/education-policy-committee>

Reform of the K–12 education system at any level or subject is notoriously difficult.

fied many of the issues outlined in this column. Two key parts of its agenda are: First, educate policymakers and national groups on the importance of the field of computing, the value of teaching computer science, and that computer science curricula focus is on conceptual knowledge. Second, ensure that rigorous computer science classes count toward a student's core graduation requirements in math or science areas.

- ▶ Realizing that reform begins with the states, the Computer Science Teachers Association (CSTA) has formed a cohort of master teachers in many states to serve as a network for sharing information and communicating issues to state and local education leaders.

- ▶ The single point of national leverage for computer science reform is the current Advanced Placement (AP) computer science course. The AP system is run by the College Board, which is a national non-profit organization that helps set the curriculum and writes and administers the tests. Every student taking an AP course is, or should be, exposed to the same curriculum, and every student takes the same test to assess the knowledge. The National Science Foundation is currently funding a project to create a new rigorous and engaging Advanced Placement Computer Science (AP CS) course that will attract more college-bound students.

- ▶ Some states are moving toward allowing the AP CS course to count toward a credit in students' core courses requirements. Texas now allows AP CS to count as a mathematics credit as part of the student's four-year mathematics requirement. North Carolina and Ohio also are moving toward allowing AP CS to count as a mathematics credit.

- ▶ Georgia has launched its Georgia

Computes! program. As part of this effort Georgia overhauled its state standards for computing and now counts AP CS as a science credit.

- ▶ The National Center for Women and Information Technology (NCWIT) has formed the K–12 alliance, bringing together leaders from more than 20 organizations with a potential reach of approximately half the girls in the U.S. to provide resources and advocate for reform.

- ▶ As noted in a previous issue of *Communications*,^h recognizing the need for a rigorous and engaging computing course taken before AP, Joanna Goode and Jane Margolis are working with a team to create an exciting new curriculum based on the ACM and CSTA model curriculum for K–12 education.ⁱ This course is being scaled up in the Los Angeles school district, and they are working with the state to build it into their higher-education requirements.

These are all promising signs, but any education reform, whether it is STEM broadly or computing specifically, will have to be measured over years, if not decades. Progress can be made, but the states will have to lead, with federal policymakers offering support with resources and a heightened sense of national urgency. There is a significant risk that computing will be left on the outside looking in. The community needs to come together and use the existing efforts mentioned in this column as a starting point for reform. We need to work together to prove why rigorous and engaging computing education should be included in the K–12 landscape. In short, we need to ensure computer science education is part of the national dialogue of what students need to learn in the high-technology and highly competitive global economy. ■

h Reprogramming college preparatory computer science. *Commun. ACM* 51, 11 (Nov. 2008), 31–33.

i The model curriculum can be found here: <http://www.csta.acm.org/Curriculum/sub/AC-MK12CSModel.html>

Cameron Wilson (wilson_c@hq.acm.org) is the director of the ACM U.S. Public Policy Office in Washington, D.C.

Peter Harsha (harsha@cra.org) is the director of government affairs at the Computing Research Association (CRA) in Washington, D.C.

Copyright held by author.

Viewpoint

Face the Inevitable, Embrace Parallelism

Hardware, software, and applications must all evolve in anticipation of the proliferation of parallelism.

I TOOK MY first parallel programming class in graduate school, approximately 20 years ago. Having attended a fairly well-funded and well-equipped graduate program, my fellow students and I had at our disposal a shared-memory multiprocessor system we could use for experiments. Moreover, we had access to a Cray supercomputer at a nearby su-

percomputing center. The tools available at the time for parallel programming amounted to auto-vectorizing compilers, some vendor-specific performance libraries, and non-standard threading application programming interfaces. Much of the hardware we used was relatively unstable and the tools were buggy. It was a daunting task, though we were merely asked to develop paral-

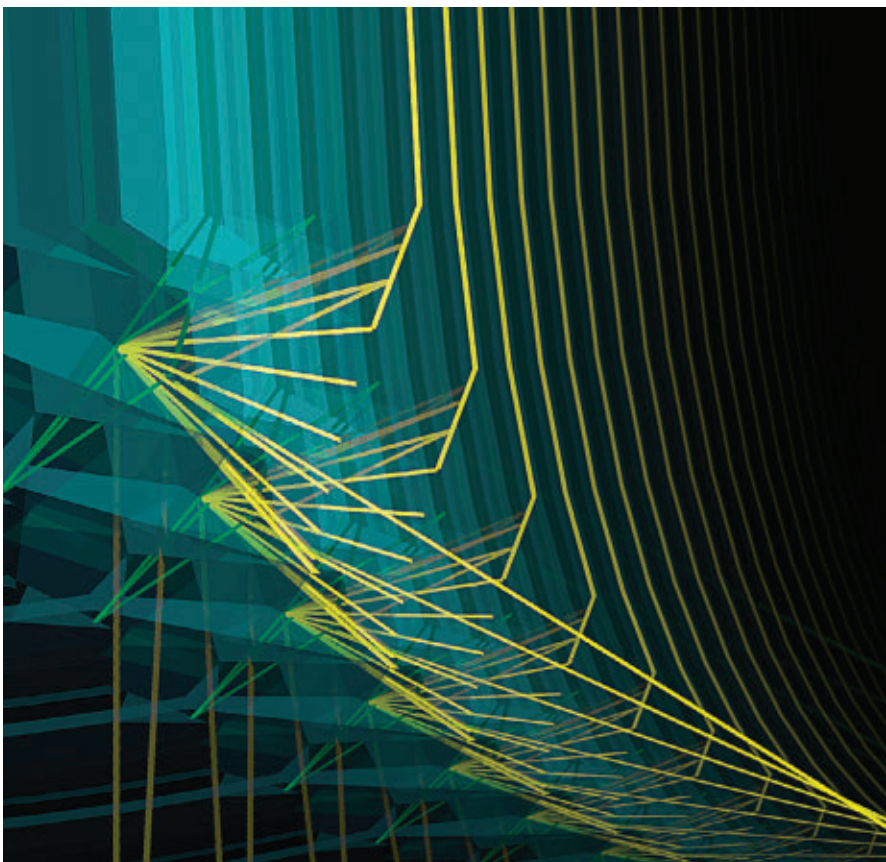
lel versions of simple algorithms, like sorts. The zeitgeist was one of experimentation with immature programming models on cold-war-fueled supercomputer innovation.

How things have changed. While some of the functional components we experimented with were quite large, they really did not approach the vast complexities faced by today's developers. With the exception of the rarified air of research labs (academic or otherwise), sequential microprocessors comprised the vast majority of computers that were used. However, in the two decades since, mainstream microprocessors have abruptly evolved to become increasingly (overtly) parallel devices.

How Software Has Changed

Calling modern applications complex does not quite do justice to the state of affairs. It may surprise some readers to learn that hardware vendors often have insight into the architectures of a broad range of applications. For Intel, this spans a fairly large variety of market segments. The reason, of course, is that hardware platform companies have a vested interest in ensuring software runs well (running best is the goal) on their products.

Applications that span hundreds of thousands or millions of lines of code are the norm. The use of externally sourced libraries is fairly commonplace. Many application frameworks are implemented at such a level of abstrac-



tion that the effective control paths and data dependencies are, for all intents and purposes, opaque to compilers used to build the applications (an interesting analysis is presented in Mitchell et al.³). Applications often simultaneously deploy what is viewed as distinct functional idioms in application development. Manipulation of dynamic and persistent databases, functionality implemented via remote servers or peers, event-based programming, graphical processing, other compute-intensive processing (of many flavors), and more occur in many modern client applications (that is, applications you might be running on your laptop).

Moreover, multi-language development has become far less surprising for the past two decades. Driven recently by Web-based application development (and most likely to be continued by parallel software development), the last 15 years or so have seen new programming languages successfully adopted commercially at a rate of approximately one every 18 months. Consider Java, C#, JavaScript, PHP, Perl, Python, Ruby, Erlang, OCaml, and Haskell, to name a few. Call it a Moore's Law analogy for programming tools (driven this time by the economics of software development rather than silicon manufacturing). This may also surprise many, especially C or C++ developers.

Productivity is one of the primary drivers of software development technology, usually even more of a factor than performance. For many market segments, time to market or deployment is the biggest influencer of tool use. Productivity is also another way to quantify development costs. It is be-

The trend toward increasing software abstraction and heterogeneity to manage complexity is accelerating.

cause of this emphasis on productivity that we are not at a plateau; instead, the trend toward increasing software abstraction and heterogeneity to manage complexity is accelerating.

And that is the challenge that hardware and software developers face in the age of mainstream highly parallel processing. Before diving into that, I will take a diversion into the trends driving increasing (software-exposed) parallelism in hardware.

How Hardware Is Changing

The oft-cited Moore's Law has been remarkably accurate at predicting the macro trends in transistor scaling on silicon manufacturing processes for the past four decades. In a nutshell, transistor densities double every two years, indirectly to a doubling of performance for the same power budget. For the most part, this performance improvement was manifested for single-threaded applications running on microprocessors. In this era, new features in microprocessors were evaluated by how effectively they used area in return for performance.

However, in the last five years or so, semiconductor manufacturers ran into a power wall, which was essentially a slowing of the power-scaling trend. In another nutshell, whereas power density per area of silicon was roughly flat from generation to generation, power was increasing somewhat. There are several physical reasons for this (some discussed in Borkar¹), but one manifestation was a slowing of voltage scaling trends (factoring in frequency, power is effectively cubic in voltage). So, in this era, new performance-oriented features in microprocessors are largely evaluated by how they use the power budget for performance.

Examples of power-efficient performance features are simpler cores that increase IPC (instructions per cycle), utilize less power-intensive techniques for latency hiding like simultaneous multithreading (rather than deeper, more speculative execution), wider execution payload per instruction (as in vector instructions, like Intel's Streaming SIMD Extensions or SSE), and more cores per die. (To be sure, microprocessor vendors are adding features that enable many other important features, like improved manageability, virtual-

Calendar of Events

September 15–18

MobileHCI '09: 11th
International Conference on Human Computer Interaction with Mobile Devices and Services,
Bonn, Germany,
Contact: Reinhard Oppermann,
Email: reinhard.oppermann@fit.fraunhofer.de

September 16–18

ACM Symposium on Document Engineering,
Munich, Germany,
Contact: Uwe M. Borghoff,
Email: uwe.borghoff@unibw.de

September 21–22

International Conference on Automotive User Interfaces and Interactive Vehicular Applications,
Essen, Germany,
Contact: Albrecht Schmidt,
Email: albrecht.schmidt.2006@gmail.com

September 21–23

Performance Metrics for Intelligent Systems,
Gaithersburg, MD,
Contact: Elena R Messina,
Email: Elena.messina@nist.gov

September 25–26

2009 Information Security Curriculum Development Conference,
Kennesaw, GA,
Contact: Michael Whitman,
Email: mwhitman@kennesaw.edu

September 30–October 2

ACM Symposium on Applied Perception in Graphics and Visualization,
Chania, Crete,
Contact: Katerina Mania,
Email: k.mania@ced.tuc.gr

September 30–October 2

MindTrek 2009,
Tampere Finland,
Contact: Artur R. Lugmayr,
Email: artur.lugmayr@tut.fi

September 30–October 2

European Conference on Cognitive Ergonomics,
Helsinki, Finland,
Contact: Leena Norros,
Email: leena.norros@vti.fi

ization, and so on.) The basic impact of these performance features is to use software-exposed parallelism to drive much of the performance improvement. For example, not utilizing SSE instruction and multiple cores in a quad-core microprocessor leaves over 90% of the peak floating point performance on the floor. Simply stated: *the trend to software-exposed parallelism is also accelerating (at the exponential pace of Moore's Law).*

Facing Parallelism

For the last three decades, software development has largely evolved to improve productivity while hardware development has largely evolved to transparently deliver sustained performance improvements to this software. This has resulted in a divergence that must be reconciled. Many of the productivity-driven software development trends are either at odds with hardware performance trends or are outpacing the abilities of tools and various frameworks to adapt. If this seems like a very hardware-centric way to look at things, I will restate this from a software developer's perspective: microprocessor architecture is evolving in a direction that existing software components will be hard-pressed to leverage.

This may seem like an overly bleak outlook; it is intended to be a reality check. It is almost certain that software developers will adapt to parallelism incrementally. It is also a certainty that the physics of semiconductor manufacturing is unlikely to change in the coming years.

I have been on both sides of the discussion between software and hardware vendors. Software vendors demand improved performance for their applications through hardware and tool enhancements (aka "the free lunch"). Hardware vendors ask software vendors to make somewhat risky (from a productivity and adoption point of view) efforts to tap new performance opportunities.

Most of the time, a middle path between these perspectives is taken wherein software vendors incrementally adopt performance features while hardware vendors enhance tools and provide consultative engineering support to ensure this happens. The end result is/will be that the path to a com-

Parallel programming in the mainstream is relatively new, while many of the tools and accumulated knowledge were informed by niche uses.

plete refactoring of their applications will take a longer, more gradual road. This middle road may be all you can hope for applications that do not evolve significantly in terms of either usage modes and data intensiveness.

However, for many applications there should be a long list of features that are enabled or enhanced by the parallelism in hardware. For those developers, there is a better, though still risky, option: *Embrace parallelism now and architect your software (and the components used therein) to anticipate very high degrees of parallelism.*

Embracing Parallelism Is Important to Software Developers

The software architect and engineering manager need only look at published hardware roadmaps and extrapolate forward a few years to justify this. Examining Intel's public roadmap alone, we get a sense of what is happening very concretely. The width of vector instructions is going from 128 bits (SSE), to 256 bits (AVX²), to 512 bits (Larrabee⁴) within a two-year period. The richness of the vector instructions is increasing significantly, as well. The core counts have gone from one to two to four to six within a couple of years. And, two- and four-way simultaneous multithreading is back. Considering a quad-core processor that has been shipping for over a year, the combined software-exposed parallelism (in the form of multiple cores and SSE instructions) is 16-way (in terms of single-precision floating-point operations) in a single-socket system (for example, desktop personal computers). Leaving this performance

"on the floor" diminishes a key ingredient in the cycle of hardware and software improvements and upgrades that has so greatly benefitted the computing ecosystem.

There is another rationalization I have observed in certain market segments. In some application spaces, performance translates more directly to perceived end-user experience today. Gaming is a good example of this. Increasing model complexity, graphics resolution, better game-play (influenced by game AI), and improved physical simulation are often enabled directly by increasing performance headroom in the platform. In application domains like this, the first-movers often enjoy a significant advantage over their competitors (thus rationalizing the risk).

I do not mean to absolve hardware and tool vendors from responsibility. But, as I previously mentioned, hardware vendors tend to understand the requirements from the examples that software developers provide. Tool vendors are actively working to provide parallel programming tools. However, this work is somewhat hampered by history. Parallel programming in the mainstream is relatively new, while many of the tools and accumulated knowledge were informed by niche uses. In fact, the usage models (for example, scientific computing) that drive parallel computing are not all that different from the programs I was looking at 20 years ago in that parallel programming class. Re-architecting software now for scalability onto (what appears to be) a highly parallel processor roadmap for the foreseeable future will accelerate the assistance that hardware and tool vendors can provide. ■

References

1. Borkar, S. Design challenges of technology scaling. *IEEE Micro* (Mar.-Apr. 2006), 58-66.
2. Intel. Intel AVX (April 2, 2008); <http://softwareprojects.intel.com/avx/>.
3. Mitchell, N., Sevitsky, G., and Srinivasan, H. *The Diary of a Datum: Modeling Runtime Complexity in Framework-Based Applications*. (2007); http://domino.research.ibm.com/comm/research_people.nsf/pages/sevitsky_pubs.html/SFILE/diary%20talk.pdf.
4. Seiler, L. et al. Larrabee: A many-core x86 architecture for visual computing. In *Proceedings of ACM SIGGRAPH 2008*.

Anwar Ghuloum (anwar.ghuloum@intel.com) is a principal engineer in Intel Corporation's Software and Services Group, Santa Clara, CA.

Copyright held by author.

Interview

An Interview with Maurice Wilkes

Maurice Wilkes, the designer and builder of the EDSAC—the first computer with an internally stored program—reflects on his career.

PRESENTED HERE ARE excerpts from an interview with Sir Maurice Vincent Wilkes, the developer of the Electronic Display Storage Automatic Calculator (EDSAC), microprogramming, symbolic labels, macros, and subroutine libraries. Wilkes, the 1967 ACM A.M Turing Award recipient and winner of the ACM lifetime membership award, is a former member of Olivetti's Research Strategy Board and an emeritus professor at the University of Cambridge Computer Laboratory in the U.K. David P. Anderson, Principal Lecturer in the History of Computing at the School of Creative Technologies, University of Portsmouth, U.K., conducted the interview with Wilkes, 96, earlier this year.

When did you first get involved with computers?

Well, you've got to realize that although there were no digital computers in the immediate pre-war period, there was a lot of digital computing. The importance and power of it was beginning to be recognized.

The actual computing was then on desk machines with people to work them, mostly research students, but professional computers were beginning to be employed for organizations such as the army, for calculating range tables or firing tables as they were called in America. That was all beginning to grow up. Cambridge was a very lively example of this digital computing.



Who was leading that activity?

We had here [John] Lennard-Jones^a who was a great pioneer of structural chemistry. And he and his small group of very able people showed that in spite of the computational bottleneck you could, in fact, achieve quite significant results. Lennard-Jones was a man of much vision and he was successful persuading the university to establish a computing laboratory, which was initially called a mathematical laboratory. It was Lennard-Jones who gave me my first opportunity to get practical experience of computing.

a Sir John Edward Lennard-Jones (1894–1954)

What was your role?

The university took me on as the boy who did the work! Analog computers were much in the air then and a differential analyzer was ordered. We were starting up this mathematical laboratory when I received an invitation to join in the war effort working on radar. Of course, I didn't know the exact nature of the work at the time of the invitation but I was one of a small group of people from the Cavendish who were let into the secret.

Who was it that told you?

It was [Robert] Watson-Watt^b himself at the Air Ministry. So, I went off to do that, deserting Lennard-Jones very ungratefully because he'd got it all fixed up.

How did Lennard-Jones react to losing you?

He didn't mind—I went off anyway. When I came back after the war, in September 1945, I found myself temporarily, but later permanently, head of the Mathematical Laboratory.

How much latitude did you have in deciding the priorities of the laboratory?

As head of the laboratory I didn't have to ask people if I could do things. The overall terms of reference were to develop mathematical methods and equipment for doing computation. So that was all fine. As I had been doing

b Sir Robert Alexander Watson-Watt (1892–1973)

radar I didn't know anything about what was going on with mathematical machines but I soon began to learn.

Did you have any help with your education in computing machinery?

I learned a great deal from [Douglas] Hartree^c who was in touch with American people and one day I had a telegram out of the blue from the Moore School at Philadelphia asking me to go to a course. I got there with great difficulty as crossing the Atlantic in those days was no simple matter. I missed the early part of the course.

Did that cause you any serious difficulties?

No, I had got a singular set of qualifications because I had done some computing as a student. I was one of the people that worked a hand-operated calculating machine. I was a thoroughly qualified electronics person having done ham radio and all that sort of thing. I had the mathematical background insofar as it was necessary for computing.

[John Presper] Eckert^d and [John] Mauchly^e were the instructors and they put me absolutely and fully in the picture. I heard them talking about stored-program computers—people say the Von Neumann^f computer, it's really the Eckert-Von Neumann computer and I thought I might have a shot at building one.

Was that the first time that you had encountered the notion of

c Douglas Rayner Hartree (1897–1958)

d John Presper Eckert Jr. (1919–1995)

e John William Mauchly (1907–1980)

f John von Neumann (1903–1957)

Although there were no digital computers in the immediate pre-war period, there was a lot of digital computing.

the stored-program computer?

No, John Von Neumann wrote a report on behalf of the group and [Leslie] Comrie^g was given a copy in America and he showed it to me. He lent it to me and I sat up all night reading it, so it wasn't the first time.

How did Comrie come to have a copy of the report?

They gave copies away to people who visited. Comrie's copy is now in the library of the Computer Laboratory.

What did you do next?

The first thing to do was to make sure an ultrasonic memory would work and we did that by January 1947 and then we went ahead.

This was quite a departure from the pre-war work of the laboratory.

Did you need any special permission to start this work?

Cambridge is a very strange place, there are little departments like mine, and big ones like the Cavendish. But from the administrative point of view they are on a level. That meant I didn't have to ask anybody or make any proposals. I was able to just go ahead and do it. There were some funds that went with the lab in effect and I guessed that more funds would become available in due course.

Did you have a large staff at your disposal?

No, no; very small. Most projects—in industry and university—depend on a small handful of three or four people and we had less than that to provide the drive. There were a lot of people who were paid on the funds, mathematicians and other hangers-on and there were also a number of assistants. We had instrument makers and electronic people on the assistant level. But I was the one who brought all the information about computers into it so there was no argument with me you see; it all came from me. I had a very loyal team and so we went ahead.

How long was it before you achieved some success?

The EDSAC began to work in the summer of 1949 on May 6th. That was the

g Leslie John Comrie (1893–1950)

day we did the first program and we'd all got little programs ready to run.

How was computer development viewed at Cambridge?

Oh I don't know, I always like to make a joke and say that they thought we were mad and if, at a cocktail party, you enlarged on your enthusiasm you would find people moving away from you!

You see, I never tried to do any proselytizing, I simply built a computer.

Did you have a clear sense from the start of who your users were likely to be?

They were all around me, they were students who didn't like spending weeks, or a week, or more computing. They rushed at a computer, even an unsatisfactory experimental one, as EDSAC was to begin with. And it was through those students that the idea spread. They went to their supervisors and said "Look what I've done." The supervisors were duly impressed and before very long important people in Cambridge were saying that computers were important. It was very low-key, bottom-up, students-upward. That's not a bad way for ideas to spread.

Did you have any concerns about how the computer-building work of the laboratory would be funded going forward?

Well, I assumed it would all happen. We were a very low-cost outfit because we didn't have a lot of the mathematicians and people on the payroll for the sake of the money and I was in 100% charge, which made it very easy.

Am I correct in thinking that the initial capital budget or the laboratory in 1936 was around £10,000?

Yes.

That was a very large sum at the time.

It was. Lennard-Jones was a man of enormous vision and although analog computers were in the air the laboratory was not biased toward analog computers. We could drop them as soon as it appeared that they didn't work out and I could go ahead and build a stored-program computer.

That must have been a very liberating environment in which to work?

Very. Yes, it was a very responsible one. I mean no one else in the laboratory was sure that it was going to work but I was the one who could see it first. And so you see, we then had an enormous advantage. This is what really gave us the edge. Because quite a number of these computers, especially the one in Manchester, were beginning to work at the same time. But they all had to hire an engineer to build a computer for them but that wasn't the case with me.

I was fully qualified on both sides. I got a group of students working on programming before the computer was running and so we could make a very quick and rapid transition to the user side and that was where we got the edge.

Could you say a little about the different contributions mathematicians and engineers made to early attempts to build computers? Was there any tension between them?

Well, of course. Mathematicians weren't particularly well qualified. They'd all done a bit of numerical analysis but it wasn't the same as digital computing. I think perhaps tension arose from entirely different backgrounds. Take the question of Boolean algebra. Mathematicians often write and speak as though Boolean algebra and mathematical logic was at the basis of computing but it wasn't that way at all. The mathematicians did not understand switching really, electronic switching; the engineers did.

Mathematicians, when it was pointed out to them, that Boolean algebra modeled electronic switching at once understood and because they could understand digital switching to a certain degree by understanding mathematical logic, they assumed that everyone would look at it that way.

Whereas engineers, when they were first told about Boolean algebra, thought "What a daft idea this all is!" and it was only later when Shannon told them about the connection that they saw any use for Boolean algebra.

But there wasn't any use. Boolean algebra has no time element to it and while it is good for shaking up a bit of



M.V. Wilkes during EDSAC I construction; EDSAC I became operational in 1949.

complex logic we didn't have complex logic. We all had very simple logic in the early days. Eckert is on record somewhere saying that he looked at Boolean algebra but it didn't seem to him to be useful. None of the practical people made much use of Boolean algebra but it was regarded as absolutely essential to the mathematicians. But there was a tension between them that is perfectly true.

Did that give rise to any problems at Cambridge or elsewhere?

Of course so many of the physicists of the period had been through the mathematical tripos that was one of its strengths. But not all of them, many Cavendish people and supervisors like [John Ashworth] Ratcliffe^h had no understanding of mathematics.

I was ensconced in the four walls of a computer laboratory and I never counted myself as a mathematician.

Von Neumann, of course, rather despised engineers. He got on with them all right but I don't think he regarded them as important for such matters as having credit for what they were doing.

[Alan] Turingⁱ was an exact contemporary of mine and that means that I don't have to regard him as a great man because you don't regard your contemporaries as great men. I don't remember him very clearly from the undergraduate days but he was certainly in the class and we took the tripos togeth-

er and we both got the highest honors you could. So that was all right.

He was a real mathematician except that he only learned one little bit of mathematics and then didn't learn any more. He was no practical organizer and, well, if you had Turing around in the place you wouldn't get it going.

That certainly wasn't a problem with the EDSAC.

No, I mean we just barged ahead on the EDSAC and the rule was that if you had got something that would work you didn't spend another hour on making it simpler or cheaper, you went ahead with it.

It demanded very strict discipline and keeping your eye on the ball. There were all sorts of interesting things to follow up but we resisted them.

We concentrated on the one objective with no demonstrations on the way. We didn't need to show that the ultrasonic memory would work. I mean when the electronics end, it was working that was sufficient. Whereas, you see, at Manchester they had an electrostatic storage depending on quantum theory and they had to be very sure that it would work. It was [Tom] Kilburn's^j idea to build a 'Baby'. He was able to do it. Validating the memory was what the Baby was all about. It was absolutely essential because they had to validate it not for themselves but for their sponsors.

I wasn't troubled with sponsors. Somehow the money came.

To what extent would you say that the work of [Charles] Babbage^k was significant in shaping the early development of stored-program computers?

I didn't know anything about Babbage. People started writing letters to the *Times* and Hartree got interested and I remember him coming into our building with a copy of Babbage's memoirs in his hand. It was Hartree who got me interested in Babbage. Of course, Babbage never had the concept of the stored program, instructions being coded as numbers; Babbage certainly wasn't influencing me.

^h John Ashworth Ratcliffe (1902–1987)

ⁱ Alan Mathison Turing (1912–1954)

^j Tom Kilburn (1921–2001)

^k Charles Babbage, FRS (1791–1871)

Looking back, what would you say was the significance of Turing's 1936 Entscheidungsproblem paper?

I always felt people liked to make a song and dance. Something like the doctrine of the Trinity involved whereas to an engineer you've only got to be told about the stored program idea and you'd say at once "That's absolutely first-rate, that's the way to do it." That was all there was to know.

There was no distinction in that paper that had any practical significance. He was lucky to get it published at all but I'm very glad he did. I mean [Alonzo] Church^l had got the same result by other methods.

I liked Turing; I mean we got on very well together. He liked to lay down the law and that didn't endear him to me but he and I got on quite well. People sometimes say I didn't get on with Turing but it's just not true. But then I was very careful not to get involved.

Was he a difficult man with whom to get along?

Yes, I think that's probably true and he was not in any sense a team leader. He didn't know how to get things done.

Of course I had another advantage there. I had war service, six years of it, and I had done real staff jobs and that teaches you a lot about how to get things done. [Max] Newman^m was a great admirer of Turing. But he was not in the line management of the computing work; I mean Newman was never an engineer. The professor of electrical engineering did that.

[Freddie] Williams?ⁿ

Yes, everybody at TRE [Telecommunications Research Establishment] had some experience of management. Williams was in charge of the computer at Manchester and he was a very strong-minded person. Mind you he was a leader too—he ran it like a dictator!

You can't design or build a computer unless you're an engineer. I mean that's what you mean by being an engineer. Newman exerted very little influence on what went on in Manchester. Williams saw to that all right.

^l Alonzo Church (1903–1995)

^m Maxwell Herman Alexander Newman (1897–1984)

ⁿ Sir Frederic Calland Williams (1911–1977)

Did Newman's involvement with the Colossus have any effect on developments at Manchester do you think?

No, I don't think Williams would have been interested in the technology because, as I say, when technology moves, it moves very fast. And the technology that was used in the Colossus was very different from the sort of technology that took root in Bawdsey [radar station].

Was there any rivalry between the various computer-building projects about who would get there first?

Well, as I always say, it was a funny race because we were all aiming at different finishing points. You see, we wanted something that was business-like and would fit into this existing digital environment. Eckert and Mauchly wanted to produce a commercially viable computer and I don't quite know what Williams wanted to do. He had no permanent interest in computers. He wasn't very interested in computers at all. He was interested in showing that CRT memory would work but I don't think he had any interest beyond that and he handed it over to Kilburn who made very good use of it. Kilburn was a very, very great success.

What are your recollections of Kilburn?

I knew him very well. Of course we were very good friends and we were both determined that we wouldn't allow any Manchester-Cambridge rivalries to show up in our groups and we achieved that on the whole, I mean we always had a high respect for each other. It could so easily have happened, you know. But it didn't and that was due to Kilburn's common sense really and mine. It was very important but I mean we were complimentary.

What was Kilburn's interest in computers?

He was interested in providing a computing service as I was.

Returning to Newman for a moment. We now know that in 1945, Newman took Willis-Jackson, who was William's predecessor, to Bletchley Park to see the Colossus. Did Newman ever discuss the Colossus with you—

even in the most general terms?

No, and I don't think Williams would have been interested in the technology because, as I say, when technology moves, it moves very fast. And the technology that was used in the Colossus was very different from the sort of technology that took root in Bawdsey.

Another important figure at Manchester at that time was Patrick Blackett.^o Did you have anything to do with Blackett?

Blackett? Oh, I knew Blackett, he was always very nice to me. He said he didn't know anything about computers and that was perfectly true. He was of an age. Above a certain age people are never really happy with computers.

He helped Williams and Newman though?

Well, he was a busybody so he would be everywhere. He was very energetic and he knew how to get things done. He thought socialism was a great thing, whereas I thought socialism was a great mistake and indeed it was.

What have you found most surprising about the developments that have taken place in computers since 1949?

Well, of course, it's the speed. We had great vision, we saw that computers were going to be a big thing, not only for arithmetic calculation but in other things as well, business and whatnot. We had great vision but we could have no idea of timescale. For one thing, young men don't but the other reason was of course we couldn't see the coming of semiconductors. Now semiconductors have given us various things, small size, small cost, high power but the important thing they have given us is reliability. We used to pray for reliability—our prayer was answered. In my lectures on this sort of thing I say that it was St. Theresa who was credited with the remark that it is prayers that are answered that create more problems than those that aren't! □

^o Lord Patrick Maynard Stuart Blackett (1897–1974)



Group Term Life Insurance**

10- or 20-Year Group Term Life Insurance*

Group Disability Income Insurance*

Group Accidental Death & Dismemberment Insurance*

Group Catastrophic Major Medical Insurance*

Group Dental Plan*

Long-Term Care Plan

Major Medical Insurance

Short-Term Medical Plan***

Who has time to think about insurance?

Today, it's likely you're busier than ever. So, the last thing you probably have on your mind is whether or not you are properly insured.

But in about the same time it takes to enjoy a cup of coffee, you can learn more about your ACM-sponsored group insurance program — a special member benefit that can help provide you financial security at economical group rates.

Take just a few minutes today to make sure you're properly insured.

Call Marsh Affinity Group Services at 1-800-503-9230 or visit www.personal-plans.com/acm.

3132851 35648 (7/07) © Seabury & Smith, Inc. 2007

The plans are subject to the terms, conditions, exclusions and limitations of the group policy. For costs and complete details of coverage, contact the plan administrator. Coverage may vary and may not be available in all states.

*Underwritten by The United States Life Insurance Company in the City of New York, a member company of American International Group, Inc.

**Underwritten by American General Assurance Company, a member company of American International Group, Inc.

***Coverage is available through Assurant Health and underwritten by Time Insurance Company.

AG5217

MARSH

Affinity Group Services
a service of Seabury & Smith

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

The Google Web Toolkit is an end-run around Web development obstacles.

BY BRUCE JOHNSON

Reveling in Constraints

THE WEB'S TRAJECTORY toward interactivity, which began with humble snippets of JavaScript used to validate HTML forms, has really started to accelerate of late. A new breed of Web applications is starting to emerge that sports increasingly interactive user interfaces based on direct manipulations of the browser document object model (DOM) via ever-increasing amounts of JavaScript. Google Wave, publicly demonstrated for the first time in May 2009 at the Google I/O Developer Conference in San Francisco, exemplifies this new style of Web application. Instead of being implemented as a sequence of individual HTML "pages" rendered by the server, Wave might be described as a client/server application in which the client is a browser executing a JavaScript application, while the server is "the cloud."

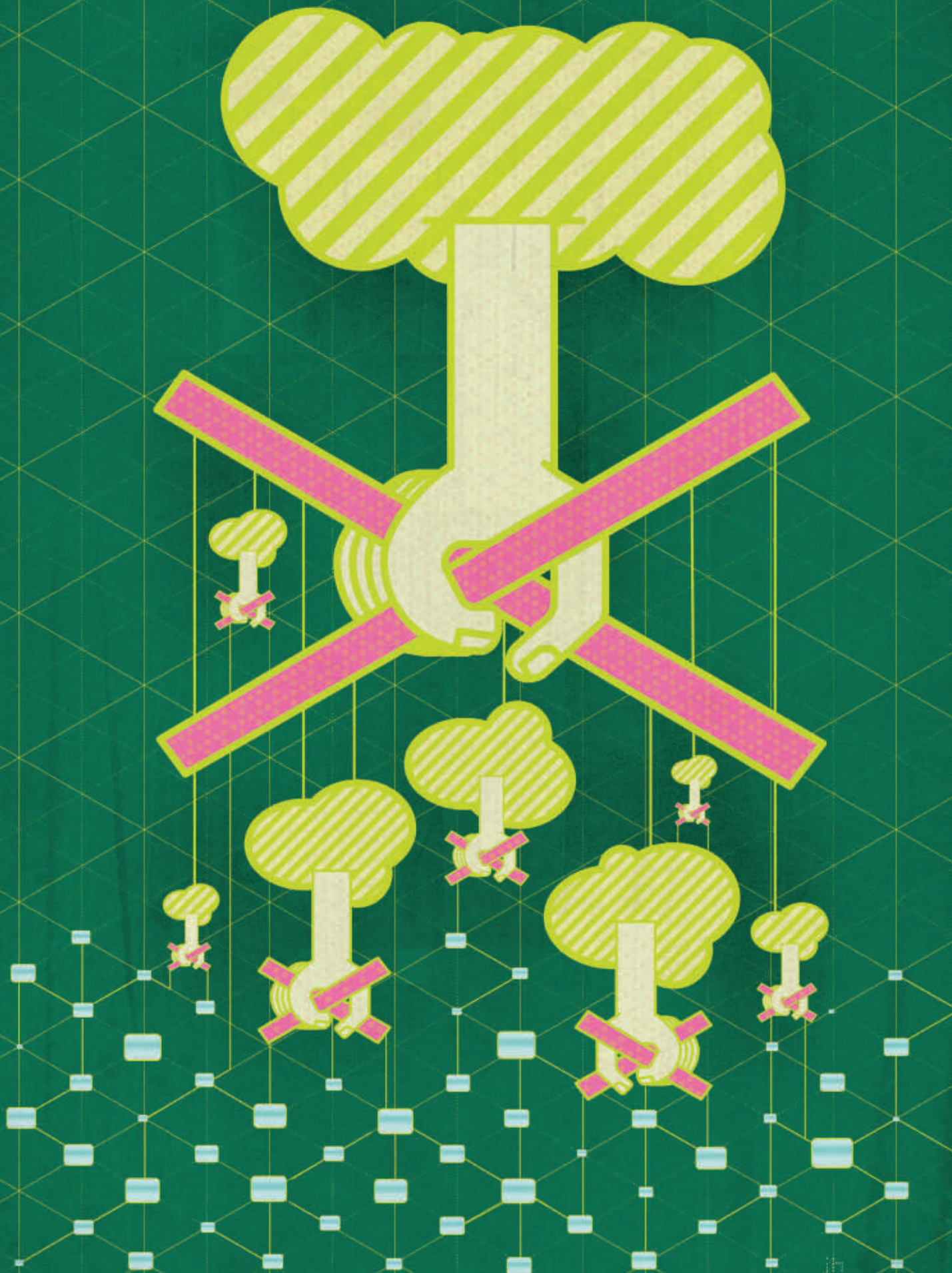
The key browser technologies responsible for enabling this new generation of Web applications are not especially new: JavaScript runs within the

browser to manipulate the browser DOM as a means for actually rendering the UI and responding to user events; CSS (cascading style sheets) are used to control the visual style of the UI; and the XHR (XmlHttpRequest) subsystem allows JavaScript application code to communicate asynchronously with a Web server without requiring a full-page refresh, thus making incremental UI updates possible. There are many more browser technologies that read like alphabet soup: XML, VML, SVG, JSON, XHTML, DTD ... the list goes on.

Curiously, these browser technologies have been available for many years, yet it has taken until now for mainstream developers to cobble them together to create compellingly interactive Web applications. Why? The opinion of the Google Web Toolkit team—a perspective that can, of course, be endlessly debated—is that the primary obstacle is literally the implementation details. It is simply too difficult to code them all to work together in a way that provides quick and reliable performance on the wide range of browsers available.

Our response was to design Google Web Toolkit (GWT) to allow developers to spend most of their time writing and debugging application code using the Java language rather than JavaScript. Working in Java means developers can leverage the productivity of Java IDEs (integrated development environments). Once they are pleased with their Java code, developers can use GWT's cross-compiler to convert Java source code into functionally equivalent, and optimized, JavaScript. The idea of cross-compilation tends to raise eyebrows, and we've heard more than our fair share of incredulity about this, so let's take a step back to describe how we decided on this approach—and how things have actually worked out.

GWT began life as a prototype that Google software engineer Joel Webber and I produced as a way to address what might best be described as the



over-constrained problem of Web development. Thanks largely to the success of Google Maps and Gmail, several points had simultaneously become clear to us:

- ▶ End users really liked and wanted browser-based applications.

- ▶ Rich client-side interactivity (for example, Maps and Gmail) made such applications much more responsive and usable than typical Web 1.0 page-at-a-time applications and thus much more compelling.

- ▶ Each major browser was technically capable of enabling such interactive applications. They could all run JavaScript and support dynamic HTML, but bugs, inconsistencies, and proprietary APIs and behaviors prevented any single JavaScript program from working consistently and efficiently on the majority of the modern browsers.

- ▶ Support for the JavaScript language itself—that is, for the “pure” language syntax and core JS libraries, excluding DOM APIs—was, to our surprise, quite consistent and reliable across browsers.

In other words, the browser—in particular, XHR, JavaScript, and the DOM—presented a capable, albeit frustrating, platform for delivering applications.

JavaScript Reservations

At the same time, we had questions about whether JavaScript was a good language in which to write business-critical applications. On the one hand, JavaScript is a flexible, dynamically typed language that makes certain types of code easy to write and pleasantly succinct. On the other hand, that same flexibility can make JavaScript harder to use within a team environment because there is no easy way to enforce the use of consistent conventions automatically across an entire codebase. It is true that, with significant extra work, a JavaScript team could insist that all script be augmented with extra metadata (JSDoc, for example) and then use additional tools to verify that all script complies with the agreed-upon conventions. This would also necessarily restrict the developers to a statically analyzable subset of JavaScript, since some of JavaScript’s most dynamic constructs—`eval()` and the `with` statement—are good exam-

ples—thoroughly defeat static analysis. All this extra stuff—the metadata and verification tools—seemed an awful lot like an ad-hoc static type system and a compiler front end.

Furthermore, we badly wanted an IDE. Our experience had thoroughly convinced us that IDEs are a boon to productivity, quality, and maintenance. Features that are status quo in modern Java IDEs such as code completion, debugging, integrated unit testing, refactoring, and syntax-aware search were virtually nonexistent for JavaScript. The reason for this, again, is related to the dynamism of

JavaScript. For example, it isn’t possible to provide sound code completion in a JavaScript editor in the general case because different runtime code paths can produce different meanings for the same symbols. Consider this legal JavaScript:

```
function foo(m) {
  alert("You called foo with
    the message: " + m); }

if (dayOfWeek() == "Wednesday") {
  foo = 3;
}
foo("Hello?"); // [1]
```

Box 1. Shape hierarchy as it might appear in Javascript (a) and Java (b).

```
function Shape() { }
Shape.prototype.getArea = function() { }

function Circle(radius) { this.radius = radius; }
Circle.prototype = new Shape();
Circle.prototype.getArea = function() { return this.radius * this.radius *
  Math.PI; }

function Square(length) { this.length = length; }
Square.prototype = new Shape();
Square.prototype.getArea = function() { return this.length * this.length; }

function displayArea(shape) { alert("The area is " + shape.getArea()); }

function runTest() {
  var shape1 = new Circle(3);
  var shape2 = new Square(2);
  displayArea(shape1);
  displayArea(shape2);
}
```

```
In Java language for use with GWT

abstract class Shape {
  public abstract double getArea();
}

class Circle extends Shape {
  private final double radius;
  public Circle(double radius) { this.radius = radius; }
  @Override public double getArea() { return radius * radius * Math.PI; }
}

class Square extends Shape {
  private final double length;
  public Square(double length) { this.length = length; }
  @Override public double getArea() { return length * length; }
}

static void displayArea(Shape shape) { Window.alert("The area is " +
  shape.getArea()); }

static void runTest() {
  Shape shape1 = new Circle(3);
  Shape shape2 = new Square(2);
  displayArea(shape1);
  displayArea(shape2);
}
```


Box 2. Implementing native Java methods in handwritten JavaScript.

```
// This is Java!
static native Element createElement() /*- {
  // This is JavaScript!
  return document.createElement("div");
} -*/;
```

The CCL knows about JSNI and rewrites it to look something like this:

```
// A static initializer is introduced in the class.
static {
  hostedBrowser.injectFunc("createElement", "return document.
createElement(\"div\");");
}

// The method becomes all-Java and is no longer native.
static Element createElement() {
  return hostedBrowser.invokeInjectedFunc(this, "createElement");
};
```

At [1], it's impossible to tell statically whether `foo` is a function or a variable, so IDE code completion can only provide “potentially correct” suggestions, which is an optimistic way of saying that you must double-check the IDE's code completion suggestions, which in turn is likely to diminish much of the would-be productivity gain to be realized from a JavaScript IDE. For similar reasons, automated refactoring tools for JavaScript are rarely seen, even while such tools are ubiquitous in the Java world. These observations made JavaScript seem less attractive as a language in which to write large applications.

We finally realized that we wanted to *develop* our source code in the Java language, yet *deploy* it as pure JavaScript. By choosing the Java language as the origination language, we could immediately leverage the great ecosystem of Java tools, especially the great Java IDEs out there. The only question was how to produce JavaScript from the Java source input. Our answer was to build a Java-to-JavaScript compiler—an optimizing compiler, in fact, because we figured that since we were going to the trouble of writing a compiler anyway, why not make sure it produced small, efficient JavaScript? Furthermore, we discovered that because Java has a static type system, it allowed for many compile-time optimizations that JavaScript—being dynamically typed—would not.

As an example of this, consider the interaction between inlining and devir-

tualization (that is, the removal of polymorphism in a method invocation). In JavaScript, developers often simulate object-oriented constructs such as polymorphism. Box 1, for example, illustrates how a simple Shape hierarchy might appear written in JavaScript and Java language for use GWT.

The source for the two examples looks nearly identical, except for minor syntax differences, the use of `@Override` (which is useful for helping to prevent bugs), and the presence of explicit type names sprinkled on fields, methods, and local variables.

Because of the extra type information, the GWT compiler is able to perform some optimizations. The unobfuscated version of the GWT compiler output looks approximately like this:

```
function runTest() {
  var shape1, shape2;
  shape1 = $Circle(new Circle(), 3);
  shape2 = $Square(new Square(), 2);
  alert('The area is ` +
shape1.radius * shape1.radius *
3.141592653589793); // [1]
  alert('The area is ` + shape2.
length * shape2.length); // [2]
}
```

Note that in [1] and [2], a cascade of optimizations was made.

First, the compiler inlined both calls to the `displayArea()` method. This proved helpful because it removed the need to generate code for that method. Indeed, `displayArea()` is completely absent from the compiled script, re-

sulting in a minor size reduction. Even better, the inlined code is amenable to being further optimized in a usage-specific context where more information is available to the optimizer.

Next, the optimizer noticed that the types of `shape1` and `shape2` could be “tightened” to types more specific than their original declaration. In other words, although `shape1` was declared to be a `Shape`, the compiler saw that it was actually a `Circle`. Similarly, the type of `shape2` was tightened to `Square`. Consequently, the calls to `getArea()` in [1] and [2] were made more specific. The former became a call to `Circle`'s `getArea()`, and the latter became a call to `Square`'s `getArea()`. Thus, all the method calls were statically bound, and all polymorphism was removed.

Finally, with all polymorphism removed, the optimizer inlined `Circle`'s `getArea()` into [1] and `Square`'s `getArea()` into [2]. Both `getArea()` methods are absent from the compiled script, having been inlined away. `Math.PI` is a compile-time constant and was also trivially inlined into [1].

The benefit of all these optimizations is speed. The script produced by the GWT compiler executes more quickly because it eliminates multiple levels of function calls.

For obvious reasons, large codebases tend to be written with an emphasis on clarity and maintainability rather than just on sheer performance. When it comes to maintainability, abstraction, reuse, and modularity are absolute cornerstones. Yet, in the previous example, maintainability and performance come into direct conflict: the inlined code is faster, yet no software engineer would write it that way. The “maintainability vs. performance” dichotomy isn't unique to Java code, of course. It is equally true that writing modular, maintainable JavaScript tends to produce slower, larger script than one would prefer. Thus, all developers building complex Web applications have to face the reality of this trade-off. The pivotal question would seem to be just how amenable your codebase is to optimization once you've written it. In that regard, the Java type system provides great leverage, and that is how the GWT compiler is able to include many optimizations

similar to the ones shown here to help mitigate the “abstraction penalty” you might otherwise end up having to pay in a well-designed object-oriented codebase.

Bringing Together Two Worlds

Of course, the creation of an environment that allows developers to build browser-based applications in Java addresses only one part of the development cycle. Like most developers, we do not produce perfect code, so we knew we would also have to address the issues involved in debugging GWT programs.

Upon first hearing about GWT, people often assume you use it in the following way:

1. Write Java source.
2. Compile to JavaScript with GWT's compiler.
3. Run and debug the JavaScript in a browser.

In fact, that is not the way you work in GWT at all. You spend most of your time in GWT's *hosted mode*, which allows you to run and debug your Java code in a normal Java debugger (for example, Eclipse), just as you're accustomed to doing. Only when the application is written and debugged do you need actually to compile it into JavaScript. Thus, everyone's reflexive fear of never being able to understand and debug the compiled JavaScript proves to be unfounded.

The secret to making hosted mode an effective debugging environment is that it does not merely *simulate* the behavior of a browser while debugging in Java. Hosted mode directly combines true Java debugging with a real browser UI and event system. Hosted mode is conceptually simple, and it executes in a single JVM (Java Virtual Machine) process:

1. Launch an instance of an actual browser, embedded in-process, that can be controlled by Java code via JNI (Java Native Interface). We call this the *hosted browser*.
2. Create a CCL (compiling class loader) to load the GWT module's entry-point classes.
3. Whenever the CCL is asked to fetch a class, it checks to see if the class has JSNI (JavaScript Native Interface) methods. If not, the class can be used directly. If native methods are found,

the class gets compiled from source and the JSNI methods are rewritten.

4. Run the bytecode of the entry-point class, which will in turn request other classes be loaded by the CCL, which repeats the process from Step 3.

Step 3, rewriting JSNI methods, is the really neat part here. JSNI is the way to implement native Java methods in handwritten JavaScript, as shown in Box 2.

Thus, the hosted-mode CCL turns JSNI methods into thunks that redirect their calls into the hosted browser's JavaScript engine, which in turn drives the real browser DOM.

From the JVM's point of view, everything described here is pure Java bytecode and can therefore be debugged normally using a Java debugger. From the developer's point of view, he or she can see the true behavior of a real browser being driven by the Java source code without it first having been cross-compiled into pure JavaScript.

Which brings up perhaps the most exciting point about hosted mode: because it works dynamically with Java code and does not depend on invoking the GWT cross-compiler (which can be slow), hosted mode is really fast. This means developers get the same kind of run/tweak/refresh behavior they enjoy whenever working directly with JavaScript.

GWT thus manages to combine the benefits of a traditional optimizing compiler with the quick development turn-around of dynamic languages. Although the compilation technology may appear complex, it is actually fairly standard fare for optimizing compilers. The real technical problems we encountered along the way revolved around our efforts to create UI libraries to simultaneously account for browser-specific quirks without compromising size or speed. In other words, we needed to supply many different implementations of UI functionality—version A for Firefox, version B for Safari, and so forth—without burdening the compiled application with the union of all the variations, thereby forcing each browser to download at least some amount of irrelevant code. Our solution is a unique mechanism we dubbed *deferred binding*, which arranges for the GWT compiler to produce not one out-

put script, but an arbitrary number of them, each optimized for a particular set of circumstances.

Each compiled output is a combination of many different implementation choices, such that each script has exactly (and only) the amount of code it requires. It's worth mentioning that in addition to dealing with browser variations, deferred binding can specialize compilations along other axes as well. For example, deferred binding is used to create per-locale specializations (for example, why should a French user have to download strings localized for English, or vice versa?). In fact, deferred binding is completely open ended, so developers can add axes of specialization based on their needs.

This approach does create a large number of compiled scripts, but we reasoned it was a welcome trade-off: you end up spending cheap server disk space on many optimized scripts, and, as a result, applications download and run more quickly, making end users happier.

In any event, our experience in developing GWT has thoroughly convinced us that there's no need to give in to the typical constraints of Web development. That is, with a bit of creativity and some dedicated effort, we now know it is indeed possible to retain the richness of more familiar development environments without compromising the experience application users are ultimately to enjoy. ■

Related articles on queue.acm.org

Case Study: Making the Move to AJAX

Jeff Norwalk

<http://queue.acm.org/detail.cfm?id=1515744>

Coding Smart: People vs. Tools

Don Seeley

<http://queue.acm.org/detail.cfm?id=945135>

Debugging AJAX in Production

Eric Shrock

<http://queue.acm.org/detail.cfm?id=1506423>

Bruce Johnson founded Google's engineering office in Atlanta, right next door to his alma mater Georgia Tech, with the goal of producing Google Web Toolkit and a number of related tools intended to make Web development more efficient, effective, and a whole lot more fun.

MonALISA developers describe how it works, the key design principles behind it, and the biggest technical challenges in building it.

BY IOSIF LEGRAND, RAMIRO VOICU, CATALIN CIRSTOIU, COSTIN GRIGORAS, LATCHEZAR BETEV, AND ALEXANDRU COSTAN

Monitoring and Control of Large Systems with MonALISA

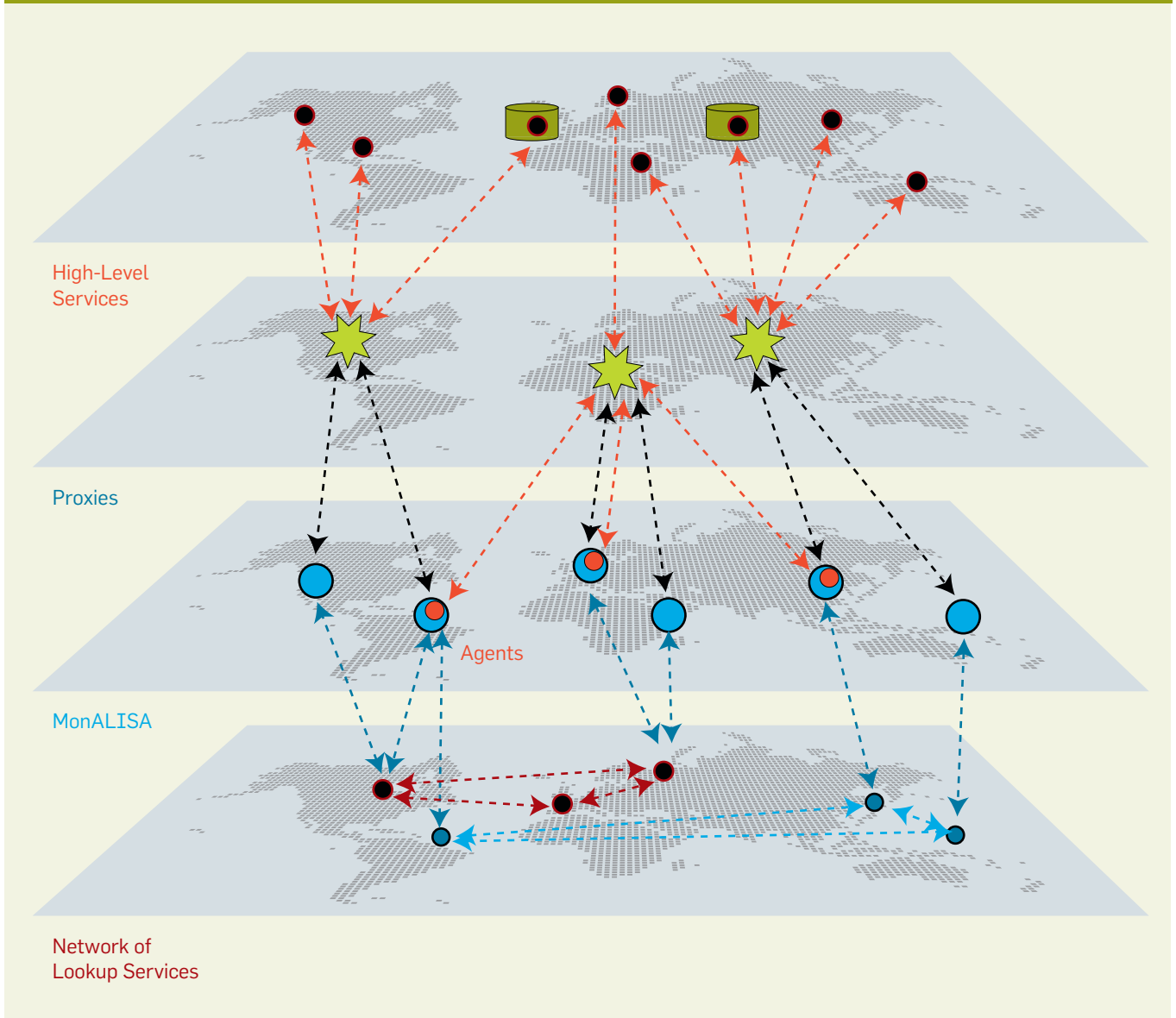
THE HIGH ENERGY physics (HEP) group at California Institute of Technology started developing the MonALISA (Monitoring Agents using a Large Integrated Services Architecture) framework in 2002, aiming to provide a distributed service system

capable of controlling and optimizing large-scale, data-intensive applications.¹¹ Its initial target field of applications is the grid systems and the networks supporting data processing and analysis for HEP collaborations. Our strategy in trying to satisfy the demands of data-intensive applications was to move to more synergetic relationships between the applications, computing, and storage facilities and the network infrastructure.

An essential part of managing large-scale, distributed data-processing facilities is a monitoring system for computing facilities, storage, net-

works, and the very large number of applications running on these systems in near real time. The monitoring information gathered for all the subsystems is essential for developing the required higher-level services—the components that provide decision support and some degree of automated decisions—and for maintaining and optimizing workflow in large-scale distributed systems. These management and global optimization functions are performed by higher-level agent-based services. Current applications of MonALISA's higher-level services include optimized dynamic

Figure 1. The four layers, main services, and components of the MonALISA framework.



routing, control, and optimization for large-scale data transfers on dedicated circuits, data-transfer scheduling, distributed job scheduling, and automated management of remote services among a large set of grid facilities.

The initial design of the MonALISA system was inspired by the Jini architecture.¹⁰ MonALISA is designed as an ensemble of autonomous self-describing agent-based subsystems that are registered as dynamic services. These services are able to collaborate and cooperate in performing a wide range of distributed information-gathering and processing tasks.

The MonALISA architecture, schematically presented in Figure 1, is based on four layers of global services. The en-

tire system is based on Java technology.⁹

The first layer is the lookup services (LUS) network that provides dynamic registration and discovery for all other services and agents. MonALISA services are able to discover each other in the distributed environment and to be discovered by interested clients. The registration uses a lease mechanism. If a service fails to renew its lease, it is removed from the LUS and a notification is sent to all the services or other applications that subscribed to such events.

The second layer of the MonALISA framework represents the network of MonALISA services. They provide the multithreaded execution engine that accommodates many monitoring modules and a variety of loosely cou-

pled agents that analyze the collected information in real time. The framework also integrates a set of existing monitoring tools and procedures to collect parameters describing computational nodes, applications, and network performance. The collected information can be stored locally in databases. Dynamically loadable agents and filters are able to process information locally and communicate with other services or agents in order to perform global optimization tasks. A service in the MonALISA framework is a component that interacts autonomously with other services either through dynamic proxies or via agents that use self-describing protocols. By using the network of lookup services,

a distributed services registry, and the discovery and notification mechanisms, the services are able to access each other seamlessly. The use of dynamic remote event subscription allows a service to register an interest in a selected set of event types, even in the absence of a notification provider at registration time.

Proxy services make up the third layer of the MonALISA framework. They provide an intelligent multiplexing of the information requested by the clients or other services and are used for reliable communication among agents. This layer can also be used for access-control enforcement to provide secure access to the collected information and the remote services management.

Higher-level services and clients access the collected information using the proxies' layer. A location-aware, load-balancing mechanism is used to allocate these services dynamically to the best proxy service. The clients, other services, or agents can get real time or historical data by using a predicate mechanism for requesting or subscribing to selected measured values. These predicates are based on regular expressions to match the attribute description of the measured values that a client is interested in. They may also be used to impose additional conditions or constraints for selecting the values. The subscription requests create a dedicated priority queue for messages. The communication with the clients is served by a pool of threads. The allocated thread performs the matching tests for all the predicates submitted by a client with the monitoring values in the data flow. The same thread is responsible for sending the selected results back to the client as compressed serialized objects.

Having independent threads for clients allows sending the information they need in a fast and reliable way, avoiding the interference caused by communication errors that may occur with other clients. In case of communication problems, these threads will try to reestablish the connection or clean up the subscriptions for a client or service that is no longer active.

Communication Lessons

One of the most difficult parts in de-



Our strategy in trying to satisfy the demands of data-intensive applications was to move to more synergetic relationships between the applications, computing, and storage facilities and the network infrastructure.



veloping the MonALISA system was the communication mechanism for all these services in the wide area network. The system tries to establish and maintain reliable communication among services, using the ability to reconnect automatically or find alternative services in case of network or hardware problems. Although, the fashion of the time was to implement remote call protocols over XML and to use Web services, we decided to use a binary protocol especially to avoid the overhead of wrapping everything in a text-based protocol and because of the lack of remote notification except for a pull-based approach (the Oasis Web Services Notification¹⁴ appeared later and still used a pull-based approach in the first implementations). Although XML or Web services still make perfect sense for certain applications, they are not appropriate for large dynamic data.

Initially we used the Java RMI (remote method invocation) as the communication protocol between clients and services. It was an elegant solution and helped us in the beginning to develop the other components of the framework without focusing too much on the underlying communication protocol. As soon as we started deploying the monitoring service on more and more sites, however, we had to replace this approach for two main reasons. The first was security concerns for the computing centers within HEP and the difficulty opening ports in the firewalls of those centers for incoming TCP connections. In some cases even the outgoing connectivity had to be restricted to a few IP addresses and ports. This was in fact the main reason for developing the layer of proxy services, allowing all the other MonALISA services to communicate with each other even when running behind firewalls or local NAT (network address translation) environments.

The second reason we had to replace RMI was because of its relatively low performance and stability in WAN connections (see Figure 2). The main operating system used in the HEP community was and still is Linux, but different flavors of it—kernels and libraries—and, of course, under a heterogeneous administration. Java

helped a lot, but we have experienced stalled sockets and poor network throughput because of the TCP stack implementation in the 2.4 kernels used at the time.

We tried to find the best balance between performance and time spent developing a custom protocol, so we still used the native Java serialization. Because of the initial aim to react in almost real time, we had to develop our keep-alive mechanism at the application level; we could not control and also had problems with the one at the kernel level. Implementing our own communication protocol over standard TCP sockets helped us to have a finer control in case of network I/O errors for quick and clean recovery. Although the TCP implementation⁵ has changed in the latest 2.6 kernels—and even though the default congestion protocol works reasonably well without any special settings—we still believe that, depending on the time constraints for the application, any remote call protocol will be an issue in WAN environments because of the intrinsic overhead combined with network latency.

At the other end, for the LAN communication between thousands of monitored entities and the local MonALISA service, we decided to take another approach: use a UDP (User Datagram Protocol)-based binary but highly portable protocol employing external data representation (XDR)¹⁵ for data encoding. This choice proved

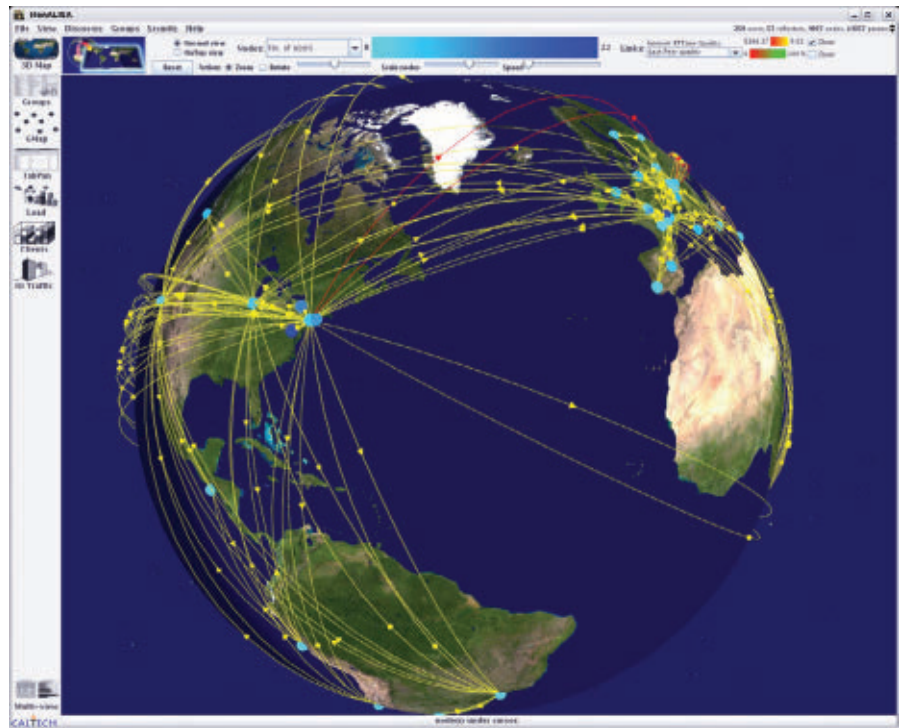


Figure 2. Monitoring the quality of WAN links.

to be effective and allowed the service to collect more than 5,000 messages per second without any loss—TCP would not have scaled to receive data simultaneously from all the nodes in a large computing farm. The ApMon client library (available in Java, C, Perl, and Python) that we developed for this purpose became the preferred way of tracing remote jobs and nodes, as it could send not only user-specific data, but also process and machine monitoring information.

Challenges of a Large, Data-Intensive Scientific Project

One of the largest communities using the MonALISA system is ALICE (A Large Ion Collider Experiment),¹ one of four LHC (Large Hadron Collider) experiments at CERN (European Organization for Nuclear Research).⁴ The ALICE collaboration, consisting of more than 1,000 members from 29 countries and 86 institutes, is strongly dependent on a distributed computing environment to perform its physics program. The ALICE experiment will start running this year and will collect data at a rate of up to four petabytes per year. During its design lifetime of 20 years, ALICE will produce more than 10^9 data files per year, and require tens of thousands of CPUs to process and analyze them. The CPU and storage capacities are distributed over more than 80 computing centers worldwide. These resources are heterogeneous in all aspects, from CPU model and count to operating system and batch queuing software. The allocated resources should increase over time to match the increase in the data-acquisition rate resulting from changes in experiment parameters, so that a doubling is foreseen in two years, and so on.

The ALICE computing model requires a dedicated node in each com-

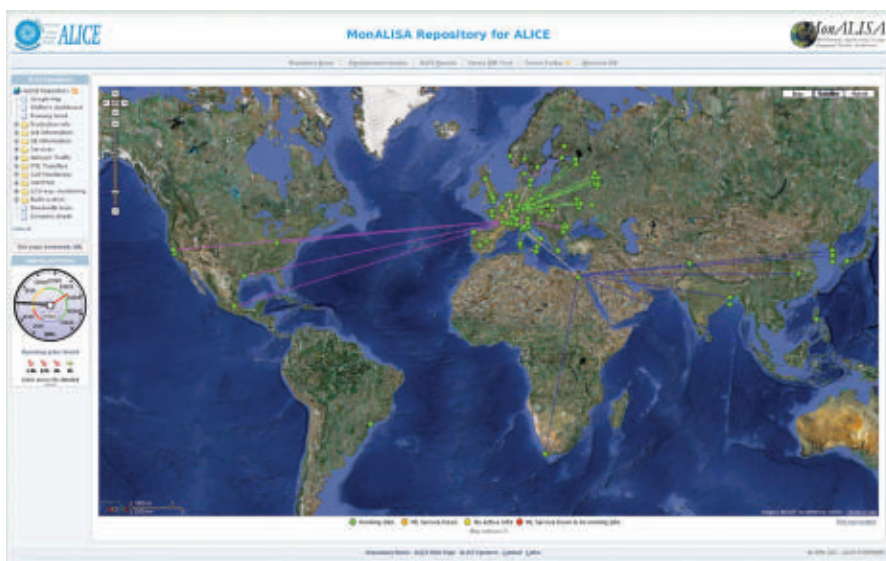


Figure 3. The MonALISA repository for ALICE. Lines represent site relations (Tier0-Tier1-Tier2).

puting center that runs the management software for the local resources. The same node is also running a MonALISA service that collects monitoring information from all computing nodes, storage systems, data-transfer applications, and software running in the local cluster. This yields more than 1.1 million parameters published in MonALISA, each with an update frequency of one minute. Moreover, ALICE-specific filters aggregate the raw parameters to produce system-overview parameters in realtime. These higher-level values are usually collected, stored, and displayed in the central MonALISA Repository for ALICE¹² (see Figure 3) and are the fuel for taking automatic actions.

In this particular case we have managed to reduce the data volume to only about 35,000 parameters by aggregating, for example, the entire CPU usage by all the jobs in the local cluster in a single parameter, by summing up network traffic on all machines, and so on. These overviews are usually enough to identify problems and take global actions in the system, and they can be stored in a central database for long-term archival and analysis. The details are available on demand on the origi-

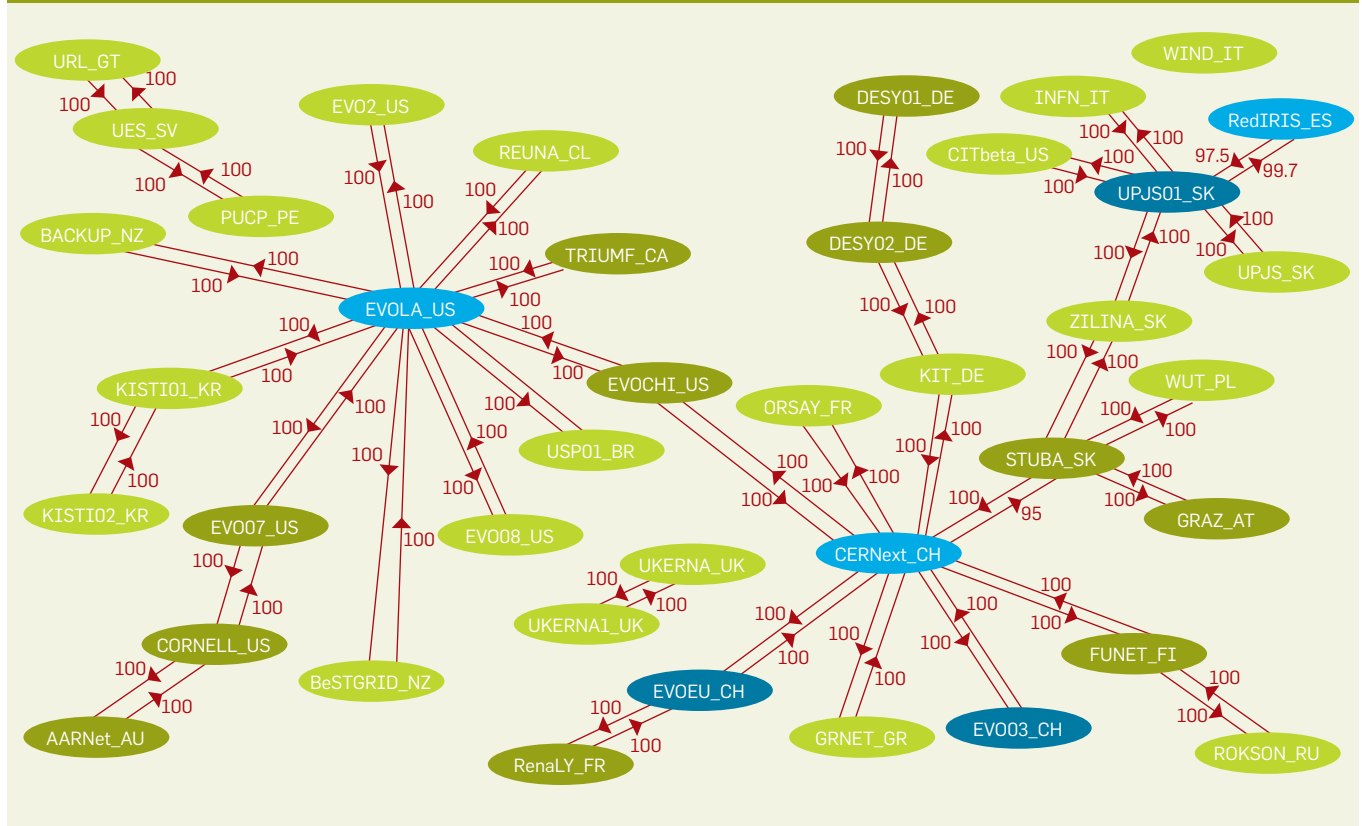
nating site and can be consulted with the GUI client. This approach proved to be very useful for debugging purposes—for example, to track the behavior of a particular application or host.

The ALICE computing model matched closely with MonALISA’s architecture so the pieces fit naturally together, but it also provided us big opportunity to fulfill the project’s initial goal: using the monitoring data to improve the observed system. Indeed, the actions framework implemented within MonALISA represents the first step toward the automation of decisions that can be made based on the monitoring information. It is worth noting that actions can be used in two key points: locally, close to the data source (in the MonALISA service) where simple actions can be taken; and globally, in a MonALISA client where the logic for triggering the action can be more sophisticated, as it can depend on several flows of data. Hence, the central client is equipped with several decision-making agents that help in operating this complex system: restarting remote services when they don’t pass the functional tests, sending e-mail alerts or instant messages when automatic restart

procedures do not fix the problems, coordinating network-bandwidth tests between pairs of remote sites, managing the DNS-based load balancing of the central machines, and automatically executing standard applications when CPU resources are idle.

The actions framework has become a key component of the ALICE grid. Apart from monitoring the state of the various grid components and alerting the appropriate people to any problems that occur during the operation, this framework is also used to automate the processes. One such automation takes care of generating Monte Carlo data that simulates the experiment’s behavior or analyzes the data. In normal cases jobs run for 10 to 12 hours and generate or analyze files on the order of 10GB each. ALICE jobs, however, can fail for a number of reasons: among the most frequent are network issues and local machine, storage, or central services problems. By continuously monitoring the central task queue for production jobs, the MonALISA repository takes action when the number of waiting jobs goes below the preset threshold (4,000 jobs at the moment). First, it looks to see whether any failed jobs can be re-

Figure 4. EVO overlay network topology—A Dynamic Minimum Spanning Tree.




scheduled to run; then if the queue length is still too short, it will schedule new bunches of 1,000 jobs. The same framework is used to copy data automatically to remote sites and to test the network connectivity among all endpoints. Combining the continuous testing of network and storages with error reporting has proved to be an efficient tool in debugging the system.

Having so many parameters to store and display on demand in a reasonably short time was a challenge—made even more difficult because the charts are generated on the fly based on users' options. Database response time depends on the number of values, so one step toward generating charts on demand was storing averaged values over increasing time intervals, saving space but losing resolution. Three database structures are filled in parallel: from one with high resolution that keeps data only for the last couple of months to one with very low resolution that keeps data forever. The data controller automatically selects which portions of data to extract from which structure to meet user requests, and it can fetch data from more than one structure if the request parameters demand it.


The second step to reduce the response time was spreading the queries to more than one database back end. Three identical database instances now receive all the updates while the select queries are split so that different parameters are fetched in parallel from all active back ends. Having these two options in place allows serving some 20,000 dynamic pages per day from a single front-end machine while the database size has reached 170GB.

Data-Transfer Services and Optimized Communication

To support large-scale data-driven applications, such as those specific to the HEP community—particularly as the amount of data becomes more prevalent—a large set of subsystems has to be configured and tuned simultaneously. Performing these operations manually not only demands expensive human expertise, but also limits the maximum practical size of such a system. Also, it becomes difficult to deal with dynamically changing conditions and errors and to coordinate the resource requirements of different



It is fair to say that at the beginning of this project we underestimated some of the potential problems in developing a large distributed system in WAN, and indeed the “eight fallacies of distributed computing” are very important lessons.



applications. Within the MonALISA framework, we developed a large set of modules and agents able to monitor different network devices, the network topology, and connectivity, and we tried to use this information in near real time to optimize the communication and data transfer in the WAN. This framework has been used for the past three years to monitor and coordinate large data transfers; we made demonstrations of the entire system at the Supercomputing conference in 2006,⁸ 2007,² and 2008.³

One example of such a system is the optimization of the global connectivity for the EVO collaboration network's videoconferencing system.⁶ The optimization is based on continuous end-to-end monitoring, including the end user's computer, as well as the network infrastructure. This way the user is informed of any potential or arising problems (for example, excessive CPU load or packet loss), and when possible, the problems are resolved automatically and transparently on the user's behalf (for example, switching to another server node in the network, reducing the number of received video streams, among others). The EVO servers communicate with each other through a set of channels—secure TCP connections—that form an overlay network on top of the actual network topology. Dedicated MonALISA services are used to collect the monitoring data from all the EVO servers and to maintain the connectivity tree (minimum spanning tree) that connects the reflectors. This tree is used to compute the optimal routes for the videoconferencing data streams dynamically, based on information about the quality of alternative possible connections between each pair of reflectors. If one or more links goes down or is substantially degraded, the tree is rebuilt and reoptimized in real time, making EVO resistant to failures (see Figure 4).

A second example in which we used MonALISA was for monitoring and controlling optical switches and providing a global service to create on-demand optical paths/trees for end-user applications.¹³ The agents use MonALISA's discovery layer to “discover” each other and then communicate among themselves autonomously, using the proxy services. Each proxy service can

handle more than 15,000 messages per second, and several such services are typically used in parallel. This ensures that communication among the agents is highly reliable, even at very high message-passing rates.

The set of agents is also used to create a global path or tree, as it knows the state and performance of each local and wide area network link, and the state of the cross connections in each switch. The routing algorithm provides global optimization by considering the “cost” of each link or cross-connect. This makes the optimization algorithm capable of being adapted to handle various policies on priorities and preservation schemes. The time to determine and construct an optical path (or a multicast tree) end to end is typically less than one second, independent of the number of links along the path and the overall length of the path. If network errors are detected, an alternative path is set up rapidly enough to avoid a TCP timeout, so that data transport continues uninterrupted.

The most laborious part of developing such global services that try to control the connectivity in the WAN is the handling of communication errors. Parts of our environment are in hybrid networks—some in research or dedicated networks only and some reachable from both academic and commercial networks. Most of the time everything works as expected and problems do not occur very frequently. When they do occur, however, it is important to understand what’s happening before acting upon it. In particular, we would like to discuss two possible cases of asymmetry in the system. When this happens only at the routing level, both sides involved in communication can reach each other, but by using different routes—this impacts the throughput and reliability of the communication, is not hard to detect, and is usually easy to recover from.

Another more serious problem occurs when different parts of the distributed framework involved in decisions have different views of the system. We had a case where some services in Europe could not reach the services in the U.S., while at the same time, some of them could see all the others. When you have a partial but consistent view of the system, you can act locally, but


in this case we reached the conclusion that the best approach was to stay on the safe side and not make any decisions. Such problems do not occur frequently in our environment, but it is really difficult to detect them and avoid making wrong decisions for the types of systems we described.

Conclusion

During the past seven years we have been developing a monitoring platform that provides the functionality to acquire, process, analyze, and create hierarchical structures for information on the fly in a large distributed environment. The system is based on principles that allow for scalability and reliability together with easing communication among the distributed entities. This approach to collecting any type of monitoring information in such a flexible distributed framework can be used in further developments to help operate and efficiently use distributed computing facilities.

It is fair to say that at the beginning of this project we underestimated some of the potential problems in developing a large distributed system in WAN, and indeed the “eight fallacies of distributed computing” are very important lessons.⁷

The distributed architecture we used, without single points of failure, proved to offer a reliable distributed service system. In round-the-clock operation over the past five years we never had a breakdown of the entire system. Replicated major services in several academic centers successfully handled major network breakdowns and outages.

As of this writing, more than 350 MonALISA services are running around the clock throughout the world. These services monitor more than 20,000 compute servers, hundreds of WAN links, and tens of thousands of concurrent jobs. Over 1.5 million parameters are monitored in near real time with an aggregate update rate of approximately 25,000 parameters per second. Global MonALISA repositories are used by many communities to aggregate information from many sites, properly organize them for the users, and keep long-term histories. During the past year, the repository system served more than 8 million user requests. 

Related articles on queue.acm.org

Monitoring, at Your Service

Bill Hoffman

<http://queue.acm.org/detail.cfm?id=1113335>

Modern Performance Monitoring

Mark Purdy

<http://queue.acm.org/detail.cfm?id=1117404>

Web Services and IT Management

Pankaj Kumar

<http://queue.acm.org/detail.cfm?id=1080876>

References

1. ALICE Collaboration; <http://aliceinfo.cern.ch/> Collaboration.
2. Caltech High Energy Physics. High Energy Physicists Set New Record for Network Data Transfer. Supercomputing 2007 (Reno); http://media.caltech.edu/press_releases/13073.
3. Caltech High Energy Physics. High Energy Physicists Team Sets New Data-Transfer World Records. Supercomputing 2008 (Austin); http://media.caltech.edu/press_releases/13216.
4. CERN; <http://www.cern.ch>.
5. Default TCP implementation in Linux 2.6 kernels; <http://netsrv.csc.ncsu.edu/twiki/bin/view/Main/BIC>.
6. EVO Collaboration Network; <http://evo.caltech.edu>.
7. Fallacies of distributed computing; http://en.wikipedia.org/wiki/Fallacies_of_Distributed_Computing.
8. HPC Wire. Physicists set record for network data transfer. Supercomputing 2006; <http://www.hpwire.com/topic/networks/17889729.html>.
9. Java; <http://java.sun.com/>.
10. Jini; <http://www.jini.org/>.
11. MonALISA; <http://monalisa.caltech.edu>.
12. MonALISA Repository for ALICE; <http://pcalimonitor.cern.ch>.
13. Voicu, R., Legrand, I., Newman, H., Dobre, C., and Tapus, N. A distributed agent system for dynamic optical path provisioning. In *Proceedings of Intelligent Systems and Agents (ISA)*, part of the IADIS Multi Conference on Computer Science and Information Systems (Lisbon, 2007).
14. WS Notification; http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsn.
15. XDR; http://en.wikipedia.org/wiki/External_Data_Representation.

Iosif Legrand is a senior research engineer at Caltech and the technical lead of the MonALISA project. He has worked for more than 16 years in high-performance computing, algorithms, modeling, and simulation, control and optimization for distributed systems.

Ramiro Voicu is a research engineer at Caltech working for USLHCNet at CERN, where he was a Marie Curie fellow. His research interests include global optimization in distributed systems and high-performance data transfers.

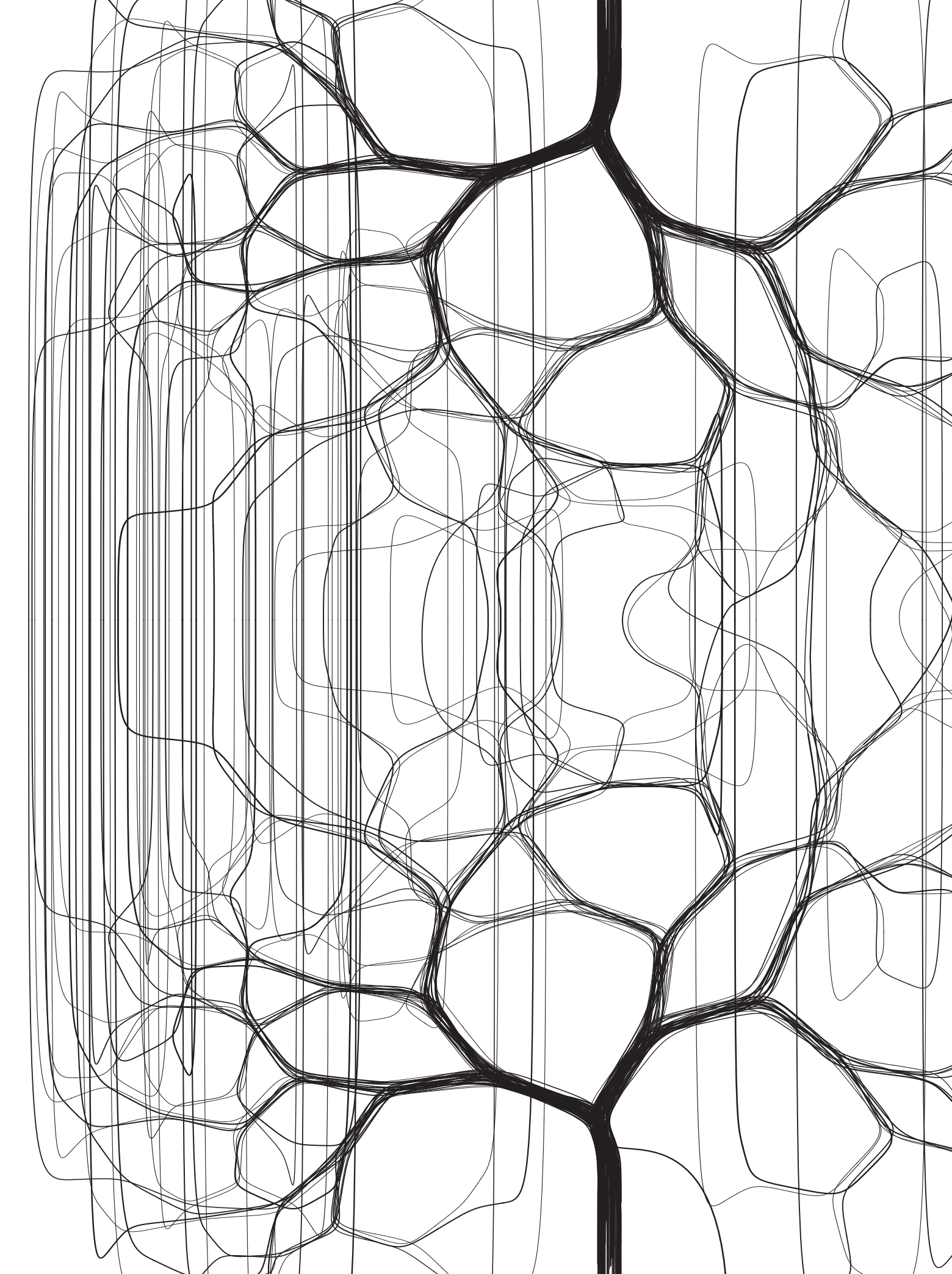
Catalin Cirstoiu is a software engineer in the finance industry. He works on parallel and distributed systems, focusing on reliability, optimizations, and high-performance issues.

Costin Grigoras is a software engineer in ALICE at CERN, where he is a fellow. His research interests include distributed systems monitoring and automated decision making.

Latchezar Betev is working in the offline team of the ALICE collaboration at CERN and is responsible for the operation of the Grid infrastructure of the experiment. His main interests include large-scale distributed computing, monitoring, and control of remote systems.

Alexandru Costan is a Ph.D. student and teaching assistant at the computer science department of the University Politehnica of Bucharest. His research interests include grid computing, data storage and modeling, and P2P systems.

© 2009 ACM 0001-0782/09/0900 \$10.00



All revision-control systems come with complicated sets of trade-offs. How do you find the best match between tool and team?

BY BRYAN O'SULLIVAN

Making Sense of Revision-Control Systems

MODERN SOFTWARE IS tremendously complicated, and the methods that teams use to manage its development reflect this complexity. Though many organizations use revision-control software to track and manage the complexity of a project as it evolves, the topic of how to make an informed choice of

revision-control tools has received scant attention. Until fairly recently, the world of revision control was moribund, so there was simply not much to say on this subject.

The past half-decade, however, has seen an explosion of creativity in revision-control software, and now the leaders of a team are faced with a bewildering array of choices.

Concurrent Versions System (CVS) was the dominant open source revision-control system for more than a decade. While it has a number of severe shortcomings, it is still in wide use as a legacy system. Subversion, which was written to supplant CVS, became popular in the mid-2000s. (Perforce is a nota-

ble commercial competitor to Subversion) Both Subversion and CVS follow the client-server model: a single central server hosts a project's metadata, and developers "check out" a limited view of this data onto the machines where they work.

In the early 2000s, several projects began to move away from the centralized development model. Of the initial crop of a half-dozen or so, the most popular today are Git and Mercurial. The distinguishing feature of these distributed tools is that they operate in a peer-to-peer manner. Every copy of a project contains all of the project's history and metadata. Developers can share changes in whatever arrangement suits their

needs, instead of through a central server.

Whether centralized or distributed, a revision-control system allows members of a team to perform a handful of core tasks:

- ▶ It allows a team to track the history of the files they work on during the development of a project. People can see who made a change; understand when and why it was made; inspect the details of the change; and re-create the state of the project at the time the change was made.

- ▶ People can work on independent subprojects without being disturbed by other people's changes and without affecting the work of their colleagues. These self-contained lines of development are usually referred to as *branches*. Branches are also used to manage the maintenance of releases that are no longer actively developed.

When the work on a subproject is complete, it can be integrated back into the larger project. This is referred to as merging.


Each revision-control tool emphasizes a distinct approach to working and collaboration. This in turn influences how a team works. As a result, no revision-control tool will suit every team: each tool comes with a complicated set of trade-offs that can be difficult even to see, much less to evaluate.

Branches and Merging: Balancing Safety and Risk


On a large project, managing concurrent development is a substantial sticking point. Developers are sadly familiar with progress on their feature being stalled by a bug in an unrelated module, so they prefer to manage this risk by working in isolated branches. When a branch is sequestered for too long, a different kind of risk arises: that of teams working in different branches making conflicting changes to the same code.

Merging changes from one branch into another can be frustrating and dangerous—one that can silently re-introduce fixed bugs or create entirely new problems. These risks can arise in several ways:

- ▶ Developers working in separate branches may modify the same sections of one or more files in different ways. A revision-control system will identify



The major difference between Subversion and the distributed tools is this: with Subversion, committing a change implicitly publishes it, whereas with the distributed tools, the two are decoupled.



these sections as conflicts that need to be resolved by hand. Whoever resolves the conflict must choose one branch's version, the other, or a hybrid.

- ▶ Code in one branch may depend on functionality that has changed in the other branch. In many cases, this dependency will be obvious: it will lead to a broken build. Sometimes the effects can be much more insidious, causing an unanticipated kind of failure.

- ▶ Some systems do not cope well if files have been renamed or copied in one branch but modified under their old names in another. (These are more often bugs than fundamental deficiencies, but longstanding bugs are important in their own right.)

Since merges introduce risk beyond the sort that normal development incurs, how a revision-control system handles both branches and merges is of great importance. Under Subversion, creating a new branch is a matter of making a copy of an existing branch, then checking out a local view of it. Although branches are relatively cheap to create, Subversion allows several developers to work concurrently in a single branch. Since working out of a single branch carries no immediately obvious costs, most teams maintain few branches.

This mode of work introduces a new risk. Suppose Alice and Bob are concurrently working on the same files in a single branch. Subversion treats the history of a branch as linear: revision 103 follows revision 102 and precedes revision 104. Alice and Bob have each checked out a copy of revision 105 of the branch from the server onto their own laptops. These working copies contain their uncommitted work, isolated from each other until one commits his or her changes.

If Alice commits her work first, it will become revision 106. Subversion will not allow Bob to commit his work as revision 107 until he has merged his work with Alice's revision 106. Since Bob cannot commit his work, what will happen if something goes wrong with his merge? He will have no permanent record of what he did and faces some scary possibilities: his work might be lost or quietly corrupted. Because Subversion offers working out of a shared branch as the path of least resistance, developers tend to do so blindly with-

out understanding the risk they face. In fact, the risks are even subtler: suppose that Alice's changes do not textually conflict with Bob's; she will not be forced to check out Bob's changes before she commits, so she can commit her changes to the server unimpeded, resulting in a new tree state that no human has ever seen or tested.

Mercurial and Git are distributed, so they lack Subversion's concept of a single central server where metadata is hosted. A repository contains a stand-alone copy of a project's complete history and a working directory that contains a snapshot of the project's files. If Alice and Bob are working together on a project, Alice might clone a copy of Bob's repository, or she could clone a copy from some server. When she commits a change, it stays local to her repository on her machine until she chooses to share it somehow. She could do this by publishing it to a server or by letting Bob pull it directly from her.

Both Mercurial and Git decouple fetching remote changes from merging them with local changes. If Bob fetches Alice's revisions, he can still commit his changes without needing to merge with hers first. When he merges afterward, he will still have a permanent record of his committed changes. If the merge runs into trouble, he will be able to recover his earlier work.

Under the distributed view of revision control, every commit is potentially a branch of its own. If Bob and Alice start from the exact same view of history, and each one makes a commit, they have already created a tiny anonymous fork in the history of the project. Neither will know about this until one pulls the other's changes in, at which point they will have to merge with them.

These tiny branches and merges are so frequent with Mercurial and Git that users of these tools look at branching and merging in a very different way from Subversion users. The parallel and branchy nature of a project's development is clearly visible in its history, making it obvious who made which changes when, and exactly which other changes theirs were based upon. Both Mercurial and Git can associate names with longer-lived lines of development (for example, "the code that will become version 2.0"), so a development

that is important enough to deserve a name can have one.

Degrees of Freedom

It is instructive to take a look at where Subversion and the distributed tools give users degrees of freedom. Subversion imposes almost no structure on the hierarchy of files and directories that it manages. It lacks the concept of a branch, beyond what it provides via the `svn copy` command. Users find branches by convention in a portion of the hierarchy where people agree that branches ought to live. By convention, a single "main line of development" is called `/trunk`, and branches live under `/branches`.

Since Subversion doesn't enforce a policy for structuring branches, it has some interesting behaviors. To perform an operation across an entire branch, you have to know where in the namespace the root of the branch is. Most Subversion commands operate only on whatever portion of the namespace they are told to. If Alice has checked out `/branches/myfeature` and runs `svn commit` in her working copy of `/branches/myfeature/deep/sub/dir`, she will commit changes only in and beneath the `deep/sub/dir` directory of the branch. An absentminded commit from the wrong directory can leave Alice thinking that everything is fine but leave her colleagues with an inconsistent, broken tree.

The `svn update` command operates in the same way: it is possible to have portions of a working copy synced up to different revisions of a branch's history. This can easily lead to a working copy looking inconsistent when in fact it is accidentally composed of fragments from different times in a branch's history.

In contrast, the distributed tools treat the entire contents of a repository as the unit to work with. If you run `git commit -a` in any directory inside a repository, it will take a snapshot of all outstanding changes. With Mercurial, `hg update` operates similarly, bringing the entire working directory up to date with respect to a specific point in history. Neither tool makes it possible to check out an inconsistent view of a branch accidentally. If you manually revert a file or directory to some specific revision, the user interfaces make this

clear by displaying the affected files as modified.

Publishing Changes

Even though Subversion does not impose a structure on projects that use branches, it suggests a convention for naming branches. Thus, Subversion users who collaborate through a central server are likely to have an easy time finding each other's projects. Both Mercurial and Git make it fairly easy to publish a read-only repository on a server, but the repository's owner has to tell other people where the repository is: it could be anywhere on the Internet, not merely a well-known location on a single server host. In addition, neither system makes read-write publishing especially easy. This is by design.

Subversion's single-server model demands that collaborators who want to share changes with other people must have write access to the shared repository, so that they may publish their changes. With Git and Mercurial, it is certainly possible to follow this centralized model, but this is a matter of convention. Users often host their repositories on their own servers or with a hosting provider. Instead of publishing their changes to a shared server, their collaborators pull changes from them and publish their own modifications elsewhere.

The major difference between Subversion and the distributed tools is this: with Subversion, committing a change implicitly publishes it, whereas with the distributed tools, the two are decoupled. Combining committing with publishing is convenient in settings where all participants have write access to the server and where everyone is always connected to the same network. Separating the two adds an extra publication step but opens up the possibilities of working offline and using novel publication techniques.

For an example of novel publication, Mercurial supports ad hoc publication of repositories over a LAN using its built-in Web server, and it supports discovery of repositories using the Bonjour protocol. This is a potent combination for rapid development settings such as a software project's sprint: just open your laptop, share your repositories, and your Wi-Fi neighbors can find and pull your changes immediately, with no

server infrastructure required.

Both the centralized and distributed approaches to publication offer trade-offs. With a small, tightly knit team that is always wired, commit-as-publish can look like an easier choice. In a more loosely organized setting—for example, where team members travel or spend a lot of time at customer sites—the decoupling of commit from publication may be a better fit.

Centralized tools can be a good fit for highly structured “rule the team with an iron fist” models of management. Access can be controlled by managers, not peers. Whole sections of the tree can be made writable or readable only by employees with specific levels of clearance. Decentralized systems don’t currently offer much here other than the ability to split sensitive data into separate repositories, which is a touch awkward.

The Pull Model of Development

Many teams begin using a distributed revision-control system in almost exactly the same way as the centralized system they are replacing. Everyone clones one of a few central repositories and pushes the changes back. This familiar model works well for getting comfortable, but it barely scratches the surface of the possible styles of interaction.

Since the distributed model emphasizes pulling changes into a local repository, it naturally fits well with a development model that favors code review. Suppose that Alice manages the repository that will become version 2.4 of her team’s software project. Bob tells her that he has some changes ready to submit and gives her the URL from which she can pull his changes. When she reads through his changes, she notices that his code doesn’t handle error conditions correctly, so she asks him to revise his work before she will accept, merge, and publish it.

Of course, a team may agree to use a “review before merge” policy with a centralized system, but the default behavior of the software is more permissive. Therefore, a team has to take explicit steps to constrain itself.

Merges, Names, and Software Archaeology

Given their backgrounds, it is no surprise that Mercurial and Git have simi-

lar approaches to merging changes, whereas Subversion does things differently.

Since merges occur so frequently with Mercurial and Git, they have well-engineered capabilities in this realm. The typical cases that trip up revision-control systems during merges are files and directories that have been renamed or deleted. Both Mercurial and Git handle renames cleanly.

Subversion’s merge machinery is complicated and fragile. For example, files that had been renamed used to disappear in merges. This severe bug has been partly addressed so that files are now renamed, but they may contain the wrong contents. It is not clear that this is really a step forward.

A subtler problem with file naming often hits cross-platform development teams. Windows, OSX, and Unix systems have different conventions for handling the case of file names (such as, different answers to the question of whether FOO.TXT is the same name as foo.txt). Mercurial outshines its competition here. It can detect—and work safely with—a case-insensitive file system that is being used on an operating system that is by default sensitive to case.

Often, a developer’s first response to receiving a new bug report will be to look through a project’s history to see what has changed recently or to annotate the source files to see who modified them and when. These operations are instantaneous with the distributed tools, because all the data is stored on a developer’s computer, but they can be slow when run against a distant or congested Subversion server. Since humans are impatient creatures, extra wait time will reduce the frequency with which these useful commands are run. This is another way in which responsiveness has a disproportionate effect on how people use their software.

A Powerful New Way to Find Bugs

Although a simple display of history is useful, it would be far more interesting to have a way of pinpointing the source of a bug automatically. Git introduced a technique to do so via the `bisect` command (which proved so useful, Mercurial acquired a `bisect` command of its own). This technique is trivial to learn: you use the `bisect` command on a revision that you know did not have

the bug, and the revision that you know does have the bug. It then checks out a revision and asks you whether that revision contains the bug; it repeats this until it identifies the revision where the bug first arose.

This is appealing to developers in part because it is easy to automate. Write a tiny script that builds your software and tests for the presence of the bug; fire off a `bisect`; then come back later and find out which revision introduced the problem, with no further manual intervention required. The other reason that `bisect` is appealing is that it operates in logarithmic time. Tell it to search a range of 1,000 revisions, and it will ask only about 10 questions. Widen the search to 10,000 revisions, and the number of questions increases to just 14.

It would be difficult to overemphasize the importance of `bisect`. Not only does it completely change the way that you find bugs, but if you routinely drive it using scripts, you’ll have effectively developed regression tests on the fly, for free. Save those tests and use them!

The wily reader will observe that searching the commit history is much easier with Subversion than with the distributed tools, since its history is much more linear. The counterpoint to this is that the `bisect` command is built into the other tools, and hence more readily available and amenable to reliable automation.

Daggy Fixes and Cherry-Picking

Once you have found a bug in a piece of software, merely fixing it is rarely enough. Suppose that your bug is several years old, and there are three versions of your software in the field that need to be patched. Each version is likely to have a “sustaining” branch where bug fixes accumulate. The problem is that although the idea of copying a fix from one branch to another is simple, the practice is not so straightforward.

Mercurial, Git, and Subversion all have the ability to cherry-pick a change from one branch and apply it to another branch. The trouble with cherry-picking is that it is very brittle. A change doesn’t just float freely in space: it has a context—dependencies on the code that surrounds it. Some of these dependencies are semantic and will cause


change to be cherry-picked cleanly but to fail later. Many dependencies are simply textual: someone went through and changed every instance of the word *banana* to *orange* in the destination branch, and a cherry-picked change that refers to bananas can no longer be applied cleanly.

The usual approach when cherry-picking fails because of a textual problem (sadly, a common occurrence) is to inspect the change by eye and reenter it by hand in a text editor. Distributed revision-control systems have come up with some powerful techniques to handle this type of problem.


Perhaps the most powerful approach is that taken by Darcs, a distributed revision-control system that is truly revolutionary in how it looks at changes. Instead of a simple chain or graph of changes, Darcs has a much more powerful theory of how changes depend on each other. This allows it to be enormously more successful at cherry-picking changes than any other distributed revision-control system. Why isn't everyone using Darcs, then? For years, it had severe performance problems that made it completely impractical. These have been addressed, to the point where it is now merely quite slow. Its more fundamental problem is that its theory is tricky to grasp, so two developers who are not immersed in Darcs lore can have trouble telling whether they have the same changes or not.

Let us return to the fold of Mercurial and Git. Since these tools offer the ability to make a commit on top of any revision, thereby spawning a tiny anonymous branch, a viable alternative to cherry-picking is as follows: use bisect to identify the revision where a bug arose; check out that revision; fix the bug; and commit the fix as a child of the revision that introduced the bug. This new change can easily be merged into any branch that had the original bug, without any sketchy cherry-picking antics required. It uses a revision-control tool's normal merge and conflict-resolution machinery, so it is far more reliable than cherry-picking (the implementation of which is almost always a series of grotesque hacks).

This technique of going back in history to fix a bug, then merging the fix into modern branches, was given the name "daggy fixes" by the authors of



Choosing a revision-control system is a question with a surprisingly small number of absolute answers. The fundamental issues to consider are what kind of data your team works with, and how you want your team members to interact.



Monotone, an influential distributed revision-control system. The fixes are called *daggy* because they take advantage of a project's history being structured as a directed acyclic graph, or dag. While this approach could be used with Subversion, its branches are heavy-weight compared with the distributed tools, making the daggy-fix method less practical. This underlines the idea that a tool's strengths will inform the techniques that its users bring to bear.

Strengths of Centralized Tools

One area where the distributed tools have trouble matching their centralized competitors is with the management of binary files, large ones in particular. Although many software disciplines have a policy of never putting binary files under the management of a revision-control system, doing so is important in some fields, such as game development and EDA (electronic design automation). For example, it is common for a single game project to version tens of gigabytes of textures, skeletons, animations, and sounds. Binary files differ from text files in usually being difficult to compress and impossible to merge. Each of these brings its own challenges.

If a moderately large binary file is stored under revision control and modified many times, the space needed to store each revision can quickly become greater than the space required for all text files combined. In a centralized system, this overhead is paid only once, on the central server. With a distributed system, each repository on every laptop will have a complete copy of that file's history. This can both ruin performance and impose an unacceptable storage cost.

When two people modify a binary file, for most file formats there is no way to tell what the differences are between their versions of the file, and it is even rarer for software to help with resolving conflicts between their respective modifications. As a way of avoiding merging binary files, centralized systems offer the ability to lock files, so that only one person can edit a file in a given branch at any time. Distributed systems cannot by their nature offer locking, so they must rely on social norms (for example, a team policy of only one person ever modifying certain kinds of files).

Relative to its distributed counter-

parts, a centralized tool will make the history of a branch appear more linear. Whether this is a strength or a weakness seems to be a matter of perspective. A more linear history is easier to understand, and so requires less revision-control sophistication from developers. On the other hand, a history containing numerous small branches and merges more accurately reflects the true history of a project and makes it clearer which project state a team member's code was based on when working. For teams that prefer to keep project history tidy, both Git and Mercurial offer `rebase` commands that can turn the chaotic history of a feature into a neater collection of logical changes, more suited to an eventual merger into a project's main branch.

Centralized tools can offer policy advantages that are more difficult to achieve with distributed tools. For example, it is possible to configure a pre-commit script that will reject an attempted commit if it introduces an automated test-suite failure. With a distributed tool, this kind of check can be put in place on a shared central server, but that cannot protect developers from sharing inadvertently broken changes with each other horizontally, from one laptop to another.

What Behaviors Does a Distributed Tool Change?

The availability of cheap local commits makes the use of a rapid-fire style of development attractive with distributed tools. Suppose Alice is partway through a complicated change and decides that she wants to speculatively refactor a piece of code. With a distributed tool, she can commit her change as is, without worrying too much whether the project is in a sane state, and try her speculative change. If that experiment fails, she can revert it and continue on her way, eventually using the `rebase` command to eliminate some of the in-progress commits she made while she figured out what she was doing.

While this style of development is certainly possible with Subversion, experience suggests that it is far more common with the distributed tools. My conjecture is that the privacy of a branch on a developer's laptop, coupled with the instantaneous responsiveness of the distributed tools, somehow com-

bine to encourage more aggressive and pervasive use of revision control.

I have observed a similar effect with merges. Because they are such bread-and-butter activities with distributed tools, in many projects they occur far more frequently than with their centralized counterparts. Although all merges require effort and incur risk, when branches merge more frequently, the merges are smaller and less perilous. Ask any seasoned developer about a long-delayed merge following a few months of isolated work, and watch the blood drain out of his or her face.

What the Future Offers

We are not by any means near the end of the road in the evolution of revision-control systems. The field has received only fitful attention from academia. Much work could be done on its formal foundations, which could lead to more powerful and safer ways for developers to work together. Alas, I know of only one notable publication on the topic in the past decade.¹ As a simple example, when merging potentially conflicting changes, almost everybody uses either three-way merging, which is decades old, or unpublished ad hoc approaches in which there is little reason to be confident.

More practically, there are plenty of advances to be made in the way that distributed tools handle large projects with deep histories, for which they are a poor fit because of the volume of data involved. For organizations that have sensitive needs around assurance and security, the centralized tools do somewhat better than the distributed ones, but both could improve substantially.


Conclusion

Choosing a revision-control system is a question with a surprisingly small number of absolute answers. The fundamental issues to consider are what kind of data your team works with, and how you want your team members to interact. If you have masses of frequently edited binary data, a distributed revision-control system may simply not suit your needs. If agility, innovation, and remote work are important to you, the distributed systems are far more likely to suit your needs; a centralized system may slow your team down in comparison.

There are also many second-order considerations. For example, firewall management may be an issue: Mercurial and Subversion work well over HTTP and with SSL (Secure Sockets Layer), but Git is unusably slow over HTTP. For security, Subversion offers access controls down to the level of individual files, but Mercurial and Git do not. For ease of learning and use, Mercurial and Subversion have simple command sets that resemble each other (easing the transition from one to the other), whereas Git exposes a potentially overwhelming amount of complexity. When it comes to integration with build tools, bug databases, and the like, all three are easily scriptable. Many software development tools already support or have plug-ins for one or more of these tools.

Given the demands of portability, simplicity, and performance, I usually choose Mercurial for new projects, but a developer or team with different needs or preferences could legitimately choose any of them and be happy in the long term. We are fortunate that it is easy to interoperate among these three systems, so experimentation with the unknown is simple and risk-free.

Acknowledgments

I would like to thank Bryan Cantrill, Eric Kow, Ben Collins-Sussman, and Brendan Cully for their feedback on drafts of this article. 

Related articles on queue.acm.org

A Conversation with Steve Bourne, Eric Allman, and Bryan Cantrill

<http://queue.acm.org/detail.cfm?id=1454460>

Distributed Development: Lessons Learned

Michael Turnlund

<http://queue.acm.org/detail.cfm?id=966801>

Kode Vicious Strikes Again

<http://queue.acm.org/detail.cfm?id=1036484>

References

1. Löh, A., Swierstra, W., Leijen, D. A principled approach to version control, 2007; <http://people.cs.uu.nl/andres/VersionControl.html>.

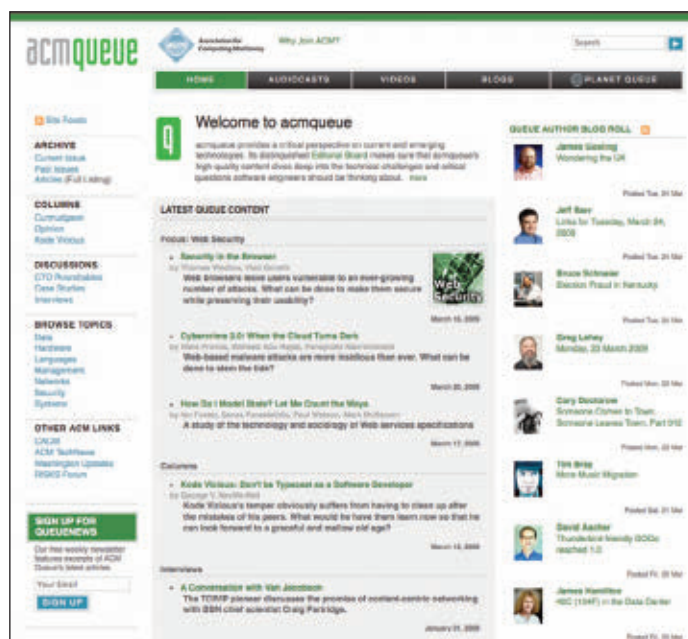
Bryan O'Sullivan is an Irish hacker and writer based in San Francisco. His interests include functional programming, HPC, and building large distributed systems. He is the author of the Jolt Award-winning *Real World Haskell* (2008) and *Mercurial: The Definitive Guide* (2009), both published by O'Reilly.



acmqueue has now moved completely online!

acmqueue is guided and written by distinguished and widely known industry experts. The newly expanded site also offers more content and unique features such as *planetqueue* blogs by *queue* authors who "unlock" important content from the ACM Digital Library and provide commentary; **videos**; downloadable **audio**; **roundtable discussions**; plus unique *acmqueue* **case studies**.

acmqueue provides a critical perspective on current and emerging technologies by bridging the worlds of journalism and peer review journals. Its distinguished Editorial Board of experts makes sure that *acmqueue's* high quality content dives deep into the technical challenges and critical questions software engineers should be thinking about.



Visit today!

<http://queue.acm.org/>

DOI:10.1145/1562164.1562185

Mining the wisdom of the online crowds generates music business intelligence, identifying what's hot and what's not.

BY VARUN BHAGWAN, TYRONE GRANDISON, AND DANIEL GRUHL

Sound Index: Charts For the People, By the People

HOW MUSIC CHARTS are created has remained relatively the same for the past 50 years despite dramatic shifts in the industry's underlying business, technological, market, and cultural assumptions. The charts, which are generated and published periodically, are based largely on retail sales and radio-listener statistics.

However, one of the most significant demographics for the industry—the teen market—has notably altered its new-music-consumption behavior due to the recent availability of online content and digital downloads. This phenomenon is recognized by chart creators eager to incorporate these observations into corporate marketing strategies in order to stay relevant to the younger generation and generate sales.

The Sound Index system demonstrates a new way to measure popularity in the world of music by

incorporating the Web, online communities, and social networks. It enables the capture of what's hot and what's not on the Web while tracking the popularity of emerging records and artists in real time. It allows the music industry to keep tabs on the demographic it considers most important and for the public to quickly learn about new music.

Music charts are useful decision-support tools that influence the visibility and success of artists, as well as help calculate their financial rewards. Popularity drives radio and television programming decisions concerning the music to be covered, the resources to be allocated, and the premiums ultimately paid to artists and their representatives. These charts are critical to the continued success of musicians, as well as music-industry professionals.

Since the late 1990s, the Web has emerged as the most popular medium for young people worldwide. Hundreds of millions of users have moved to the Web to listen to music, explore new music, and purchase individual songs, ringtones, records, and albums. In fact, 48% of teens in the U.S. did not buy a single CD in 2007, up from 38% in 2006.¹² Thus, traditional music charts are losing their relevance and appeal to their key demographics.^{15,16} Recognizing this long-term business and cultural trend, music-chart-generating organizations have begun to incorporate digital streams, but these streams still make up only a small proportion of the data reflected in the charts. In summer 2009, Apple's iTunes, which sells digital singles downloads, was the largest music retailer in the U.S. in terms of revenue.

In the U.S., Billboard (<http://www.billboard.com>) has published the Billboard Hot 100 music charts every week since 1958 (http://www.billboard.com/bbcom/charts/chart_display.jsp?g= Singles&f=The+Billboard+Hot+100). In the U.K., the British Broadcasting Corporation (BBC) has published its Top of the Pops (<http://www.bbc.co.uk/top/>) music charts since 1964. Simi-

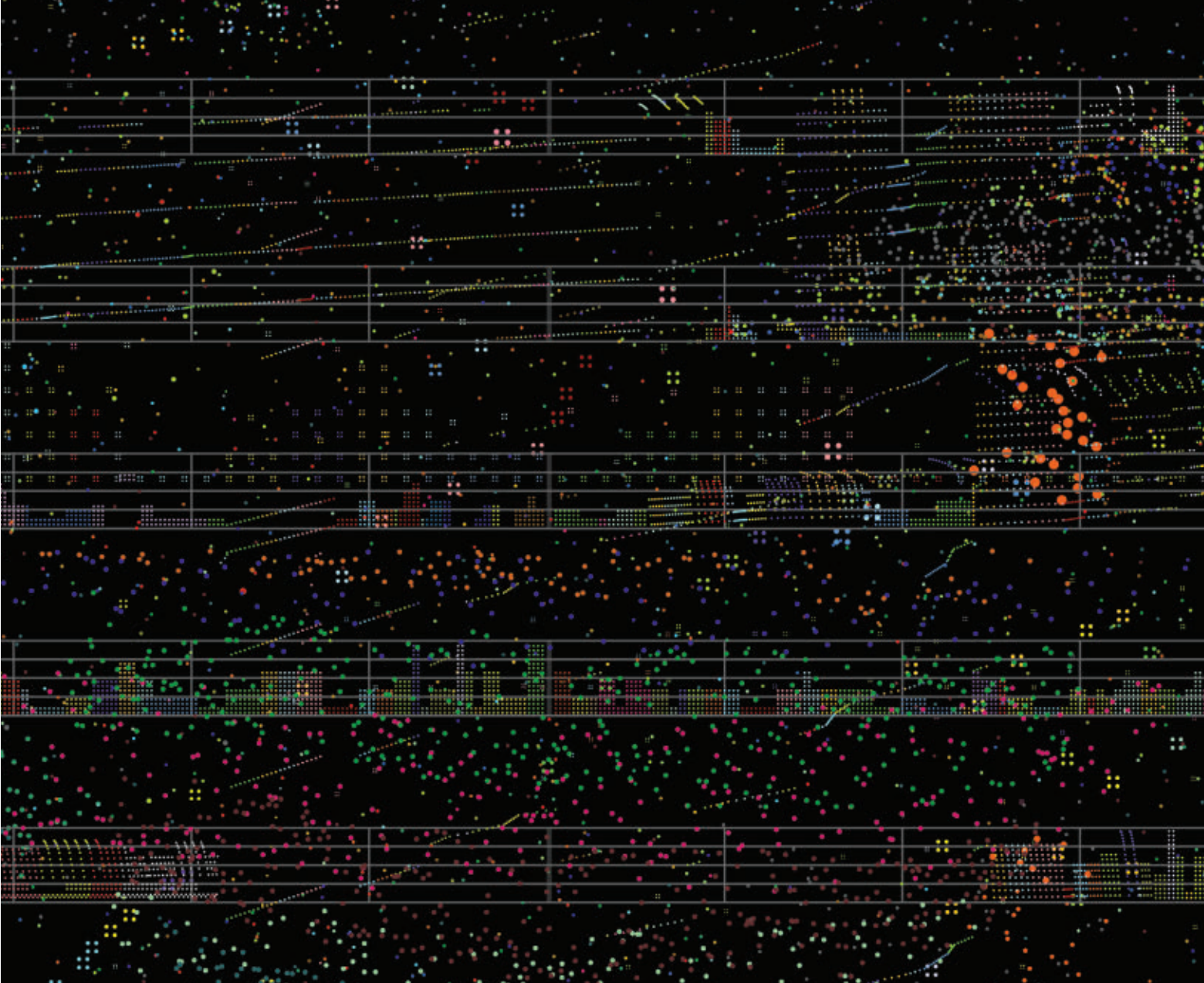


ILLUSTRATION BY STUDIO TONNE

lar charts are published in many other countries. As an exemplar, and without loss of generality, we detail here how Billboard generates its charts, highlighting the reasons for their diminishing relevance.

Traditional charts. Billboard captures data from multiple sources to produce a composite ranking of individual songs, aka singles. Its two primary sources are Nielsen Soundscan (<http://www.soundscan.com/>) and Broadcast Data Systems (<http://www.bdsonline.com/>). Soundscan tracks sales data in real time across the U.S. and Canada. Because not all retail stores have Soundscan-enabled cash registers, the data retrieved from these systems represents only a limited set of total sales. However, even this limited set is an improvement over the previous mechanism used by Billboard—mak-

ing thousands of individual telephone calls to stores across the U.S. to ask about sales.

Broadcast Data Systems collects Billboard radio-listener statistics gathered from companies contracted by Billboard to contribute to the chart of radio airplay. Thus, not all radio airplays are captured. Once the data is captured from Soundscan and Broadcast Data Systems, it is weighted by Arbitron statistics (<http://www.arbitron.com/>) and compiled by asking a random sample of the key demographic to maintain a written diary describing each radio program listened to between the hours of 6 A.M. and midnight over a period of a few months as set by Arbitron. Each diary is returned to Arbitron by postal mail; Arbitron publishes a complete set of its statistics four times per year.

In the past few years, Billboard has moved to incorporate data from digital downloads and the like, but it still constitutes only a small percentage (about 5%) of the chart's total points.¹⁰

Concerns. The music industry's desire to promote and sell new music and remove long-running singles from charts has led to the fact that the older singles that consumers are still interested in are completely ignored in the charts. Music charts also lack a clear way to handle the rerelease of singles and gauge interest in music that gains popularity over a long period through word of mouth. Another issue with the historic chart-generation process is that there is no measure for the lead-up to the release of albums or singles. Though consumers may discuss an upcoming album release for days, the charts do not reflect this conversation.

As a result, the all-time Billboard record for single-week upward movement has been broken five times since 2006.


Meanwhile, the possibility of a new payola scandal continues to haunt radio stations and record-company executives. This illegal marketing phenomenon involves record labels paying radio stations and/or disc jockeys broadcasting, and more recently streaming, records as part of a normal day's broadcast. U.S. federal law made the practice illegal in 1934, yet as of summer 2009, major record labels, including Clearchannel, CBS Radio, EMI, Sony BMG, Universal Music, and Warner Music, have come under federal investigation and in some cases had to pay tens of millions of dollars in fines and settlements. As radio airplay is a major component of the music charts and perceived popularity, these investigations in turn raise concerns about the validity of the traditional music charts themselves.

In order to address these issues and incorporate today's increasingly popular platform for music consumption, the Web, the music-charts industry must keep evolving or be left behind.


Solution

The Sound Index system catalogs the hottest artists and tracks being talked about on the Web. Incorporating "listens," plays, downloads, sales, and comments from a multitude of online communities and social networks, it provides a current view of popular music content online; the associated filtering enables customized views of the data to learn about, say, new tracks in a particular genre of interest.

The system can be divided into four distinct parts (see Figure 1), leveraging technology called MONitoring Global Online Opinions via Semantic Extraction, or MONGOOSE (<http://www.almaden.ibm.com/cs/projects/iis/mongoose/>). The first, ingestion, is the act of gathering relevant unstructured and structured content from various Web sites (such as Bebo, Google Groups, iTunes, LastFM, MySpace, and YouTube). These sources were chosen because the BBC's review team of music-domain experts identified them as relevant and important to identifying the tastes of its target demographic—teens. The system analyzes and trans-



Sound Index relies on broken-English-text analytics technology, techniques for integrating information from different modalities, and ranking technologies.



forms the data into a standard schema. The now-structured content is then stored in the system's database. Finally, the system generates music charts by applying relevant ordering schemes.

Ingestion. In an ideal world, social networking data, comments, and click streams would all have a common format that sites publish, facilitating easy download and integration of information. However, most sites lack functional application programming interfaces (APIs). As a result, screen scraping^a is the rule for data ingestion,² problematic because screen scrapers are susceptible to (even fairly minor) changes in Web sites. Unfortunately, these changes are common, as sites strive to stay fashionable in an ever-changing cultural and business environment.

Screen scrapers also require a fair amount of monitoring and maintenance. They need to log into sites and download necessary content (such as comments and view counts), transforming it into a simple format, normally just a collection of running text comments broken out (with markup removed) for further processing.

Some sites provide really simple syndication-type^b feeds that are especially useful for ingesting aggregated data (such as total listens for a particular song). Sound Index uses a combination of screen scrapers, RSS feeds, and APIs to ingest content based on the quality and reliability of each ingestion method for a given site.

Providing a reliable stream of data, even from sites that are flaky and untrustworthy, is critical to Sound Index success. As such we have developed a suite of tools and techniques to deal with common error conditions and quickly identify exotic ones and bring them to the operator's attention. In addition to the sanity-checking of values, the system monitors a number of bulk statistics on the streams themselves at each step in the processing. This monitoring allows the system to detect when, say, the quantity of documents entering the database from MySpace

a Screen scraping extracts data from machine- and display-friendly code.

b RSS is a family of Web-feed formats used to publish frequently updated works (such as blog entries, news headlines, audio, and video) in a standard format.

is, say, half of what it was yesterday. The system then spot-checks the crawler statistics; if it sees the number of documents fetched per hour has decreased, some kind of format change is likely preventing the low-level parsers from correctly splitting the comments out of the discussion pages. While these bulk statistics don't tell the operator or Sound Index itself why something is not working, they are quite effective at helping reveal when something is not working.

Sound Index automates simple corrective actions, including killing and restarting fetchers and flushing domain name system caches^c to correctly identify changes in, say, the targeted servers being crawled. Developing and automating these solutions is critical, as they reduce the need for early-morning service calls to system administrators. Sound Index uses Nagios^d to monitor all aspects of the system's performance, raising flags over problems (such as no data in the ingest feed and database-connection errors). Alba et al.² detailed additional challenges affecting Sound Index data access.

Processing. All acquired data must be "cleaned" before it undergoes processing and analysis. For example, the cleaning of structured data generally consists of a few sanity checks. For numeric data (such as total video views), which is expected to constantly increase, the system checks whether fewer total mentions were made today compared to yesterday. If they were, the implication is a negative number of views and something clearly in error.

Sound Index might report that there were zero views during this period rather than a clearly broken number for upstream processing, a scenario that is surprisingly frequent in the music domain. Also, some sources perform corrections that result in big jumps in structured numbers. As Sound Index reports data every six hours (some source numbers are updated every week), the system's developers incorporated techniques for smoothing these numbers.

A major challenge in developing the

system was figuring out how to eliminate "spam" from comment streams. Popular artists draw many visitors, a fact advertisers are quick to capitalize on. Up to 50% of a popular artist's comments are what could be considered spam (ranging from the blatant "Check out my page <URL>" to the relatively subtle "If you like this artist you will love <URL>" to the simply off topic "I like ducks!"). As they are not music-related expressions, Sound Index needs to be able to remove them from the tally; otherwise they could easily dominate (and distort) the results.

The Sound-Index topic-detection methodology accounts for whether a post is on- or off-topic, with the latter consisting of spam or nonsense posts. Employing a combination of template spotting for extremely common spam phrases and a domain dictionary, it identifies the presence or absence of music-related terminology. This approach provides reasonable spam identification, down to where it has virtually no effect on relative counts. For on-topic posts, Sound Index extracts the relevant noun phrases, as well as the associated sentiment.

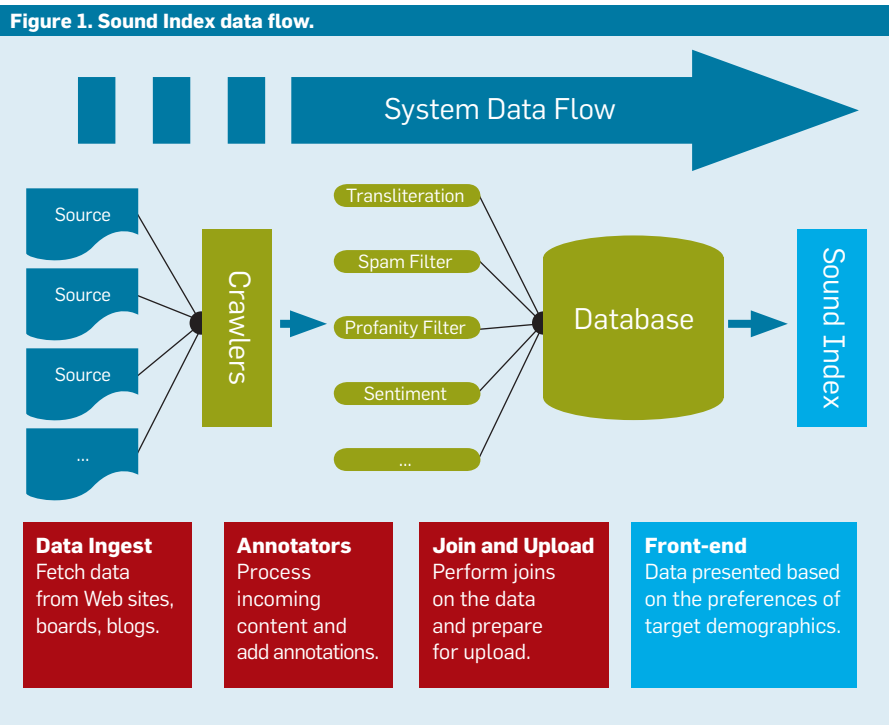
The issue of how to identify and remove spam is even more challenging due to unstructured data. Especially in the music domain, slang and nontypical spellings and linguistic constructs

appear with some frequency. A good example is the comment "U R 50 Bad." Parsing it is a complex, multi-step process. First, common variants must be rewritten into their more common English equivalents; for example, numbers as substitutions for letters must be reversed and texting abbreviations expanded. This technique results in "You are so bad" as the comment. The next step employs a feed of common slang expressions from sources like Urban Dictionary (<http://www.urbandictionary.com/>) to rewrite slang. This gets the system to "You are very good."

Sound Index must also identify ambiguous references. To do so it looks at all possible artists for "You." If it appears on a fan page for, say, Amy Winehouse, the system would conclude that she is the artist most likely being mentioned. The final parsed comment becomes "Amy Winehouse is very good," a specific mention of an artist with a positive sentiment.

The system then examines the demographic data for the poster (if available), perhaps determining that the poster is a 17-year-old female in the U.K. This data is tallied as a single mention, positive, for Amy Winehouse, by a user with said demographics. Each such data point serves as a dimension for aggregation in a subsequent step.

Resolving entity ambiguity is a ma-



c DNS is the hierarchical naming system for Internet resources; its caches help route, resolve, and link domains to IP addresses.

d Nagios (<http://www.nagios.org/>) is open source network-monitoring software.

jour challenge in chart creation. Many song titles (such as those beginning with “The”) are difficult to spot without undergoing at least shallow parsing, a task complicated by the nontypical grammatical structures often seen in the music domain. Sound Index uses a combination of context clues, domain knowledge, and poster/venue history to track “activation” of concept nodes in a domain ontology, using these activation levels to resolve the ambiguities to the greatest extent possible. This is an area of continuing research, as current implementations are simple and error-prone with more difficult resolutions, especially in cases where a band is implied by a band member with an interesting nickname (such as “The Edge” implies “U2”).

Ultimately, Sound Index converts each data element into a row of demographic data about the poster, as well as the unique ID of any track, album, or artist mentioned, along with a notion of whether the comment is positive or negative.

Data fusion for user interface generation. All data that is cleaned and assembled (into a DB2 database) must still be coalesced to create a chart, a process that is difficult in practice, as well as in theory. How does one combine mentions of an artist on a discussion board with listens from an online radio service and views of a parody of the artist’s recent video? The various methods for creating such combinations can all be viewed as a kind of “voting” of the results of different modalities and are thus amenable to examination via voting theory. To do so, the system must first enumerate the desiderata of the data-combination system. In discussion with subject-matter experts we developed several criteria for combining music-popularity data:

- ▶ Artists or tracks with broad support across the sources should do well in the ranking, reflecting “the wisdom of the crowds”;
- ▶ Artists high on one source for a day and not on other sources should not be allowed to dominate the chart.

This is a response to the common phenomenon whereby a group organizes a “flash mob” to post on the same day, usually in support of a new album to drive the band up the charts of a particular site. This anti-flooding criterion involves gaming resistance, enabling the system to handle users trying to influence or skew the charts in a particular direction;

- ▶ All sources must contribute to the final chart with no single source allowed to dominate. Thus, the disparity between counts (particularly due to differences in population size) of, say, iTunes sales and YouTube views must be reconciled; and
- ▶ The final results must be amenable to subsetting or customizable user-driven filtering; therefore, subcharts highlighting specific genera or demographics must be constructable, making it possible to produce personalized music charts.

Voting theory provides two broad classes of ways to combine these results. First is to tally the votes, perhaps through weighting; the artist or track with the most votes (plurality) is at the top of the charts. Naively counting votes is problematic, as various sources provide very different numbers; for example, sales numbers are usually much lower than views. And determining the relative importance of various modalities (such as purchases, listens, views, and posts) is subjective. Approaches like normalizing sources so their top selection is number one and weighting and combining might be the best that can be done through this approach. As long as the weights are constantly considered for changes in source popularity and the “pulsed” nature of errors in some sources is acceptable, the normalization approach reflects the important advantage of being fairly transparent. As any chart is subject to scrutiny, transparency may thus be worth the high manual cost of tracking and tuning weights.

Second is merging ranked lists, whereby each source creates a ranked list of its top- n choices. These lists are then combined without consideration of the “votes” assigned to them. For example, in Borda Counts,⁴ each #1 vote is worth n points, #2 is worth n minus 1 points, and so on. However, it suffers when n is very large and the number



Figure 2. Screenshot of the Sound Index interface from the BBC Sound Index Web site (May 7, 2008).

of voters is small, the reverse of typical elections but historically the case for music charts. In this approach, as n gets larger, the difference in effect between n and n minus 1 becomes relatively small. For this list, we found that the Nauru voting method³ (first place gets 1 point, second place $\frac{1}{2}$ point, third place $\frac{1}{3}$ point, and so on) is better at highlighting top picks. However, it is somewhat aggressive in that items ranking high on one list might also tend to dominate the overall chart. We thus introduced a variant, p , to give the system more control over this potentially skewed result. The score of an artist or track at position n thus becomes

$$\text{score}(n) = \frac{1}{p \cdot n}$$

As p varies up, that is, the system reviews entries lower on the list (such as songs at position 499 and 789) and the need for broad support becomes more pronounced. Empirical evidence suggests $p \sim 2.5$ is a good place to start.

These methods for combining data from multiple, music-related sources can be applied to full sets of data; alternatively, the initial data can be subsetted (such as to create a list of only, say, rap and hip-hop tracks) then “voted” on to create custom lists.

To evaluate this approach to combining list data, we applied, on the basis of the criteria set by the subject-matter experts, two social welfare functions:^e precision optimal aggregation¹ and Spearman Footrule distance.⁵ The former measures the number of artists from each source’s top- n list that made it to the overall top- n list; the latter emphasizes the preservation of an artist’s position in the ranking. We compared the performance of eight different methods, with performance defined as the efficacy of a given method in maximizing the two SWFs. For a detailed study of the comparison, see A. Alba. et al.³

Challenges

Sound Index is the first industrial-strength implementation of the complex idea of combining “dirty” mul-

timodal data, (see Figure 2), using unstructured information management architecture (UIMA)^{f, 6} and data mining⁷ to solve a targeted business problem. Here, we focus on two related research challenges:

Noise effects vs. freshness. Tension between the desire for frequent updates reflects the cutting edge of what is hot and the desire to minimize the influence of noise in the charts due to short-term spikes. Sound Index weighs effects (such as weekends, nights, and holidays) against events (such as new album releases, celebrity gossip coverage, and award shows). The system must ultimately compromise between being too sensitive and not reactive enough; optimizing this balance is an area for future research. For now, Sound Index employs a 24-hour window (four-to-six-hour cycle periods) to smooth out some of the effects mentioned earlier. The development team is also exploring other approaches (such as multi-month decays). Ultimately, the system needs a ranking scheme that is at least somewhat resistant to “noise” while still being able to capture freshness so, for example, it is able to identify a rise in interest in diverse sources and ignore sudden spikes in a single source.

User interface. Still unclear is the optimal way to present what is essentially an online analytical processing^g cube to end users over the Web for mining business intelligence, especially when the target audience is teens. Exploring the right way to present trending and selection is key to allowing consumers of Sound Index to get the most from the system, but doing so in a way that is obvious and intuitive is a challenge. Sound Index does offer a limited set of dimensionality tools around demographics and genres, allowing users to see charts reflecting the interests of, say, “40-something female electronica fans in the U.S.”¹⁴

Related Work

A wealth of research focuses on business intelligence mining, showcasing

the value of traditional information integration and aggregation techniques,¹⁷ whereby systems compare and contrast items with identical modalities (such as sales numbers from multiple sources). Sound Index demonstrates how to integrate information from multiple different modalities (such as comments, passive listens, sales, hits on Web sites, creation of new Web sites, and views on television), a solution required in many domains, including medical-patient preferences, drugs for certain medical conditions, cars, wine, financial products like stocks and bonds, consumer goods, cameras, computers, and books.

Nielsen’s BuzzMetrics (<http://www.nielsenbuzzmetrics.com/products>) aims for a similar goal, at least at the abstract level. Its technology monitors and analyzes consumer-generated media (such as blogs, message boards, forums, Usenet newsgroups, discussions involving email portals like Yahoo!, AOL, and MSN, opinion and review sites, and feedback and complaint sites), then analyzes, customizes, and presents the data to marketers and business-intelligence professionals, depending on client requirements. However, as of summer 2009, no publicly available technical information is available on BuzzMetrics. We speculate that its technology relies on natural-language and sentiment processing, whereas Sound Index relies on broken-English-text analytics technology, techniques for integrating information from different modalities, and ranking technologies.

Alexa Internet (<http://www.alexa.com/site/company/technology>) is another technology that crawls Web sites to produce a ranked list of sites based on traffic statistics and incoming links. It aims to generate an ordered list of the sites with the greatest volume of (incoming) traffic normally filtered by geography or other criteria, an approach that differs from the one used in Sound Index to combine data from multiple modalities into a balanced ordered list.

The effort over the past decade to address these challenges^{8,9} represents approaches to extracting and disambiguating entities within unstructured text. Sound Index faces similar chal-

e SWFs map allocations of goods and rights among people to real numbers, enabling the modeling of subjectiveness and the capture of business goals in a semiheuristic way.

f UIMA is a component software architecture that helps develop, discover, compose, and deploy multimodal analytics for unstructured information.

g OLAP is an approach to answering multidimensional analytical queries.

allenges, with disambiguation being required at the artist, band, track, and album levels.

Determining the entity being referred to in a particular text is akin to a classification problem, whereby content (“comment” in our case) must be assigned to a specific bucket, or category (artist, band, and/or track). Ellen Riloff¹³ highlighted domain-cognizant techniques for text classification; reflecting the need to focus on local linguistic context for classification and retrieval.

In terms of engineering, the world of mashups mirrors the music data requirements of Sound Index—a robust, reliable, repeatable means of gathering data from multiple, diverse online sources. ScrAPIs (Screen-scraper + API) were proposed by John Musser in 2006 as a means of mitigating the problem of unreliable or unavailable APIs from multiple content providers,¹¹ though they, too, suffer from the issues facing traditional screen-scrappers (such as breaking down when site changes are made).

Pilot

The BBC ran the Sound Index pilot from March to August 2008. Its measures for success included feedback from its editorial team, Web-use statistics, and general feedback from the online community. Despite a complete lack of marketing and promotion budget and effort, Sound Index went from a standing start as public beta in April 2008 to attract 43,469 visits from 37,900 unique users in June 2008 when it attracted 140,383 page views at an average of 3.67 per user, each spending an average of three minutes and 40 seconds on the site, or 53 seconds per page. In August 2008, it attracted more than 772,000 Web-page references.

The Sound Index team monitored the online feedback by setting up Google Alerts on all possible permutations of the project name, manually evaluating each link. There was a lot of positive comment from the Web and from the traditional business and technology press. It was named “Web 2.0 technology of the week” by the *U.K. Observer* (<http://www.guardian.co.uk/music>) for several consecutive weeks (during April to August 2008), as well

as “the hottest thing in music” (in March 2008) by the U.K.’s *Guardian Music Monthly* (<http://www.guardian.co.uk/music>). It also generated much debate in European music circles about what constitutes music popularity and what the results mean. The pilot closed August 2008, with the BBC planning for its future.

Conclusion

Called the “first definitive music chart for the Internet age,”¹⁴ Sound Index is a novel demonstration of research into processing, analyzing, collating, ranking, and presenting large quantities of unstructured and structured multimodal information in response to a change in the behavior of key demographic groups and a pressing industry need to innovate or risk being irrelevant. It is a model for demonstrating a new approach to service and product delivery, integrating (in real time) multiple, relevant online information with one’s own data to drive new and significant value for, reinvigorate connection to, and strengthen brand affinity to one’s customer base.

Here, we’ve described the system’s technical underpinnings, highlighted some of the technical challenges already addressed, and showcased the engineering and research themes that require further investigation. The underlying concepts and processes are also applicable to myriad other fields that depend on the capture of Internet buzz. We hope it inspires future software products and research projects to harness the wisdom of the crowds.

Acknowledgments

We would like to thank the BBC, specifically Geoff Goodwin, Head of BBC Switch, for its vision, support, and encouragement, as well as Alfredo Alba (IBM Almaden Research Center), Jan Pieper (IBM Almaden Research Center), Anna Liu (IBM Almaden Research Center), Bill J. Scott (formerly IBM Global Business Services), Aidan Toase (IBM Global Business Services), and IBM’s partners at NovaRising, who helped make the Sound Index system a reality. □

References

1. Adali, S., Hill, B., and Magdon-Ismail, M. The impact of ranker quality on rank-aggregation algorithms:

- Information vs. robustness. In *Proceedings of the 22nd International Conference on Data Engineering Workshops* (Atlanta, GA, Apr. 3–7). IEEE Computer Society, Washington D.C., 2006, 37.
2. Alba, A., Bhagwan, V., and Grandison, T. Accessing the deep Web: When good ideas go bad. In *Proceedings of the ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA)* (Nashville, TN, Oct. 25–29). ACM Press, New York, 2008, 815–818.
3. Alba, A., Bhagwan, V., Grace, J., Gruhl, D., Haas, K., Nagarajan, M., Pieper, J., Robson, C., and Sahoo, N. Applications of voting theory to information mashups. In *Proceedings of the Second IEEE International Conference on Semantic Computing*. (Santa Clara, CA, Aug. 4–7). IEEE Press, 2008, 10–17.
4. de Borda, J.-C. Memoire sur les elections au Scrutin. *Histoire de l’Académie Royale des Sciences 1781*; <http://asklepios.chez.com/XIX/borda.htm>.
5. Diaconis, P. and Graham, R. Spearman’s footrule as a measure of disarray. *Journal of the Royal Statistical Society, Series B (Methodological)* 39, 2 (1977), 262–268.
6. Ferrucci, D. and Lally, A. UIMA: An architectural approach to unstructured information processing in the corporate research environment. *Journal of Natural Language Engineering* 10, 3–4 (2004), 327–348.
7. Han, J. and Kambert, M. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers, Inc., San Francisco, 2001.
8. Hassell, J., Aleman-meza, B., and Arpinar, I.B. Ontology-driven automatic entity disambiguation in unstructured text. In *Proceedings of the International Semantic Web Conference LNCS 4273* (Athens, GA, Nov. 5–9). Springer, 2006, 44–57.
9. Lloyd, L., Bhagwan, V., Gruhl, D., and Tomkins, A. *Disambiguation of References to Individuals*. IBM Research Report RJ10364 (A0510-011). San Jose, CA, Oct. 28, 2005; [http://domino.watson.ibm.com/library/cyberdig.nsf/papers/D8265335C0AD4CD5852570AB00514720/\\$File/rj10364.pdf](http://domino.watson.ibm.com/library/cyberdig.nsf/papers/D8265335C0AD4CD5852570AB00514720/$File/rj10364.pdf).
10. Mayfield, G. Billboard Hot 100 to include digital streams. (July 31, 2007); http://www.billboard.com/bbcom/news/article_display.jsp?vnu_content_id=1003619084.
11. Musser, J. scrAPIs. (Mar. 21, 2006). <http://blog.programmableweb.com/2006/03/21/scrapis/>.
12. Quinn, M. and Chang, A. More teens dismissing discs in favor of online tunes. *Los Angeles Times* (Feb. 27, 2008); <http://www.latimes.com/news/nationworld/nation/la-fi-music-270208.1,2028285.story>.
13. Riloff, E. Little words can make a big difference for text classification. In *Proceedings of the 18th Annual ACM SIGIR Conference on Research and Development in Information Retrieval* (Seattle, WA, July 9–13). ACM Press, NY, 1995, 130–136.
14. Salmon, C. Click to download. *U.K. Guardian* (Apr. 18, 2008); <http://arts.guardian.co.uk/filmandmusic/story/0,,2274132,00.html>.
15. Styvén, M. *Exploring the Online Music Market: Consumer Characteristics and Value Perceptions*. Ph.D. Thesis. Department of Business Administration and Social Sciences, Luleå University of Technology, Luleå, Sweden, 2007; <http://epubl.ltu.se/14021544/2007/71/LTU-DT-0771-SE.pdf>.
16. Walsh, G., Mitchell, V.-W., Frenzel, T., and Wiedmann, K.-P. Internet-induced changes in consumer music procurement behavior: A German perspective. *Journal of Marketing Intelligence & Planning* 21, 5 (2003), 305–317.
17. Zhu, H., Siegel, M.D., and Madnick, S.E. Information aggregation: A value-added e-service. In *Proceedings of the International Conference on Technology, Policy, and Innovation: Critical Infrastructures* (The Hague, The Netherlands, June 26–29, 2001).

Varun Bhagwan (vbhagwan@us.ibm.com) is an advisory software engineer in the Computer Science Department of IBM Almaden Research Center, San Jose, CA.

Tyrone Grandison (tyroneg@us.ibm.com) is a manager in the Computer Science Department of IBM Almaden Research Center, San Jose, CA.

Daniel Gruhl (dgruhl@almaden.ibm.com) is a senior software engineer in the Computer Science Department of IBM Almaden Research Center, San Jose, CA.

Software's close encounters with the law provide some lessons for our future.

BY JAMES BOYLE

What Intellectual Property Law Should Learn from Software

TWENTY-FIVE YEARS AGO a vigorous debate raged in U.S. legal academia over whether software should be covered by patent or copyright or some third option. (Pamela Samuelson, who writes regularly in *Communications*, was co-author of the best article on the subject.⁶) In practice, software ended up

being covered by both schemes, partly due to actions by the U.S. Congress, including several references to software in the Copyright Act, and partly as a result of decisions by the Copyright Office, the Patent and Trademark Office (PTO), and by judges. One could copyright one's code and also gain a patent over the "non-obvious" novel and useful innovations inside the software. (In much of the rest of the world, software also came to be covered by copyright, though the status of patents over software was sometimes more obscure.) What can we learn from the history of the years since? A lot, it turns out, some not limited to the U.S., where intellectual property law often tends

(for better or for worse) to disproportionately influence technology policy worldwide.


At first, the use of copyright stirred the most concern. Copyright is built around an assumption of diverging innovation, the fountain or explosion of expressive activity. Different people in different situations who sit down to write a sonnet or love story, it is presumed, will produce very different results rather than be drawn to a single result. Thus, strong rights over the resulting work are not supposed to inhibit future progress. I can find my own muse, my own path to immortality. Creative expression is presumed to be largely independent of the work

of prior authors. Raw material is not needed.


There are lots of reasons to doubt that this vision of “creation out of nothing” works very well, even in the arts, the traditional domain of copyright law.⁴ But whatever its merits or defects in the arts, it seems completely wrong-headed when it comes to software. Software solutions to practical problems do converge, and programmers definitely draw upon prior lines of code. Worse still, software tends to exhibit “network effects.” Unlike my choice of novel, my choice of word-processing program is strongly influenced, perhaps dominated, by the question of what program other people choose to buy. That means that even if a programmer could find a completely different way to write a word-processing program, this programmer has to be able to make it read the dominant program’s files and mimic its features if the programmer is to attract any customers at all. This hardly sounds like completely divergent creation.

Seeing the way software failed to fit this Procrustean bed of copyright, many scholars presumed the process of forcing it into place would be catastrophic. They believed that, lacking patent’s high standards, copyright’s monopolies would proliferate. Copyright’s treatment of follow-on, or “derivative,” works would impede innovation, it was thought. The force of network effects would allow the copyright holder of whatever software became “the standard” to extract huge monopoly rents and prevent competing innovation for many years longer than the patent term. Users of programs would be locked in, unable to shift their documents, data, or skills to a competing program. Doom and gloom abounded among copyright scholars, including many who shared the premise that software should be covered by property rights. They simply believed that these were the wrong property rights to use.

Copyright did indeed cause problems for software developers, though it is difficult to judge whether they outweighed the economic benefits of encouraging software innovation, production, and distribution. But the negative effects of copyright were minimized by a remarkably prescient set of actions by courts and, to a much



For some time, the U.S. Court of Appeals for the Federal Circuit (the leading patent court in the U.S.) has seemed to believe that computers can turn an unpatentable idea into a patentable machine.



lesser extent Congress, so the worst scenarios did not come to pass. Courts interpreted the copyright over software narrowly, so it covered little beyond literal infringement. They developed a complicated test to work out whether one program infringes the details of another program.^a The details give law students headaches, but the effects were simple. If your software is similar to mine merely because it performed the same function or because I picked the most efficient way to perform some task or even because there was market demand for doing it that way, then none of those similarities counted for the purposes of infringement. Nor did material that was taken from the public domain. The result was that while someone who made literal copies of Windows Vista was clearly infringing copyright, the person who made a competing program generally would not be.

In addition, courts interpreted copyright’s fair-use doctrine to cover something called “decompilation,” basically taking apart someone else’s program so you can understand and compete with it.^b As part of the process, the decompiler had to make a copy of the program. If the law were read literally, decompilation would hardly seem a fair use. The decompiler makes a whole copy, for a commercial purpose, of a copyrighted work, precisely in order to cause harm to its market by offering a substitute good. But the courts took a broader view. The copy was a necessary part of the process of producing a competing product, rather than a piratical attempt to sell a copy of the same product. This limitation on copyright provided by fair use was needed in order to foster the innovation that copyright is supposed to encourage.

These rulings and others like them meant that software was protected by copyright but also that the copyright did not give its owner the right to prevent functional imitation and competition. Is that enough? Clearly the network effects are real. Most of us use Windows and Microsoft Word, and one very big reason is because everyone else

a See, for example, *Computer Assocs. Int’l, Inc. v. Altai, Inc.*, 982 F.2d 693 (2d Cir. 1992).

b See, for example, *Sega Enters. Ltd. v. Accolade Inc.*, 977 F.2d 1510 (9th Cir. 1992).

does. Optimists believe that the lure of capturing this huge market will keep potential competitors hungry and monopolists scared. The lumbering dominant players, goes the argument, will not become complacent about innovation or try to grab every morsel of monopoly rent. They still have to fear their raptor-like competitors lurking in the shadows. Perhaps. Or perhaps it also takes the consistent threat of antitrust enforcement. In any event, whether or not we hit the optimal point in protecting software with intellectual property rights, these rights certainly did not destroy the industry. It appeared that, even with convergent creativity and network effects, software could be crammed into the Procrustean bed of copyright without killing it off in the process. Indeed, to some, it seemed to fare quite well. They would claim that the easy legal protection provided by copyright gave a nascent industry just enough protection to encourage the investment of time, talent, and dollars, while not prohibiting the next generation of companies from building on the innovations of the past.

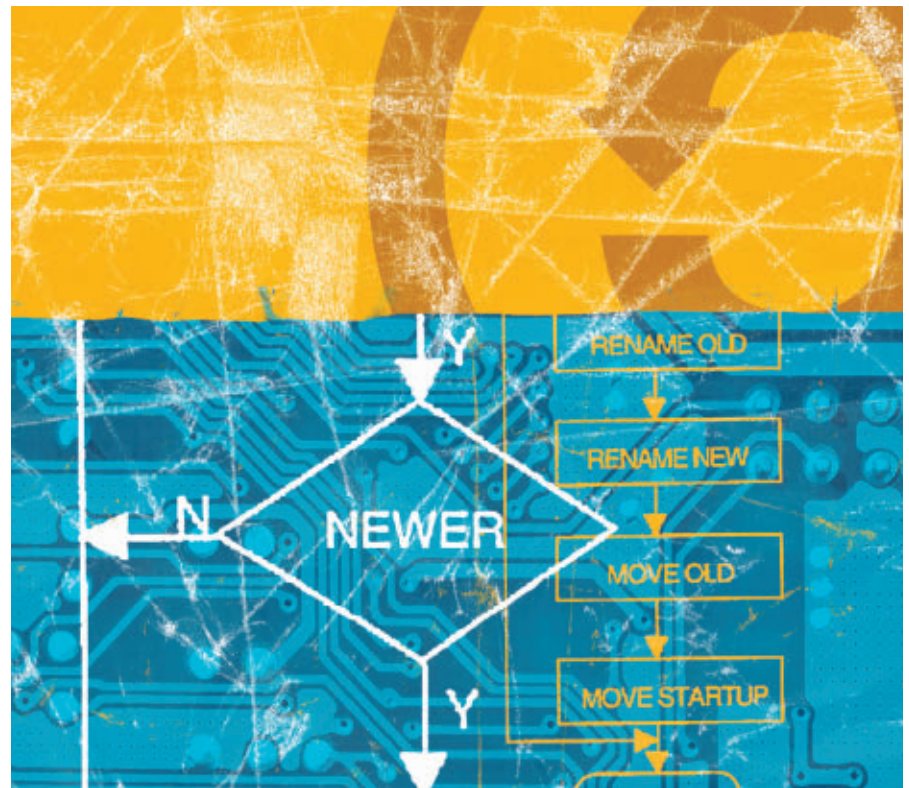
In addition, the interaction between copyright and software has produced some surprising results. There is a strong argument that it is the fact that software is copyrightable that has enabled the “commons-based creativity” of free and open source software.³ What is commons-based creativity? Basically it is creativity that builds on an open resource available to all. An additional component of some definitions is that the results of the creativity must be fed back into the commons for all to use. Think of English. You use it without license or fee, and you innovate by producing new words, slang, or phrases without clearance from some *Academie Anglaise*. After you coin your term, it is in turn available to me to build upon or use in my own sentences, novels, or jokes. And so the cycle continues. But with words we have commons-based creativity because there were no property rights over the relevant material. The software commons is different.

The creators of free and open source software were able to use the fact that software is copyrighted and that the right attaches automatically on creation and fixation to set up new distributed methods of innovation. For ex-

ample, free and open source software under the General Public License (such as Linux) is a “commons” to which all are granted access. Anyone may use the software without restriction. All are guaranteed access to the human-readable source code, rather than just the inscrutable machine code, so they can understand, tinker, and modify. Modifications can be distributed so long as the new creation is licensed under the open terms of the original. This creates a virtuous cycle whereby each addition builds on the commons and is returned to it. The copyright over the software is

have other freedoms, even if not legally guaranteed open access to source code. Still, it is difficult to deny that the extension of the property regime had—bizarrely, at first sight—actually enabled the creation of a continuing open commons. The tempting real-estate analogy would be environmentalists using strong property rights over land to guarantee conservation and open access to a green space, whereas without property rights, the space could be spoiled by all.

So much for copyright. What about patents? U.S. patent law had customar-



the “hook” that allowed software engineers to create a license that gave free access and the right to modify, and required future programmers to keep offering these freedoms. Without the copyright, those features of the license would not have been enforceable. For example, someone could have modified the open program, releasing it without the source code, thus denying future users the right to understand and modify easily. To use an analogy beloved of free-software enthusiasts, the hood of the car would be welded shut. Home repair, tinkering, customization, and re-design become practically impossible.

If there were no copyright over software at all, software engineers would

ily drawn a firm line between patentable invention and unpatentable idea, formula, or algorithm. The mousetrap could be patented, but not the formula used to calculate the speed at which it snaps shut. Ideas, algorithms, and formulae were in the public domain, as were “business methods.” Or so we thought.

The line between idea or algorithm on the one hand and patentable machine on the other looks nice and easy. But put that algorithm into a computer and things begin to look more complex. Say, for example, the algorithm was the process for converting miles into kilometers and vice versa. In the abstract, this is classic public-domain

stuff, no more patentable than $E = mc^2$ or $F = ma$. What about when those steps are put onto the tape of the Turing machine, onto a program running on the hard drive of a computer?

For some time, the U.S. Court of Appeals for the Federal Circuit (the leading patent court in the U.S.) has seemed to believe that computers can turn an unpatentable idea into a patentable machine. In fact, in this conception, the computer sitting on your desk becomes multiple patentable machines—a word-processing machine, an email machine, a machine running the pro-

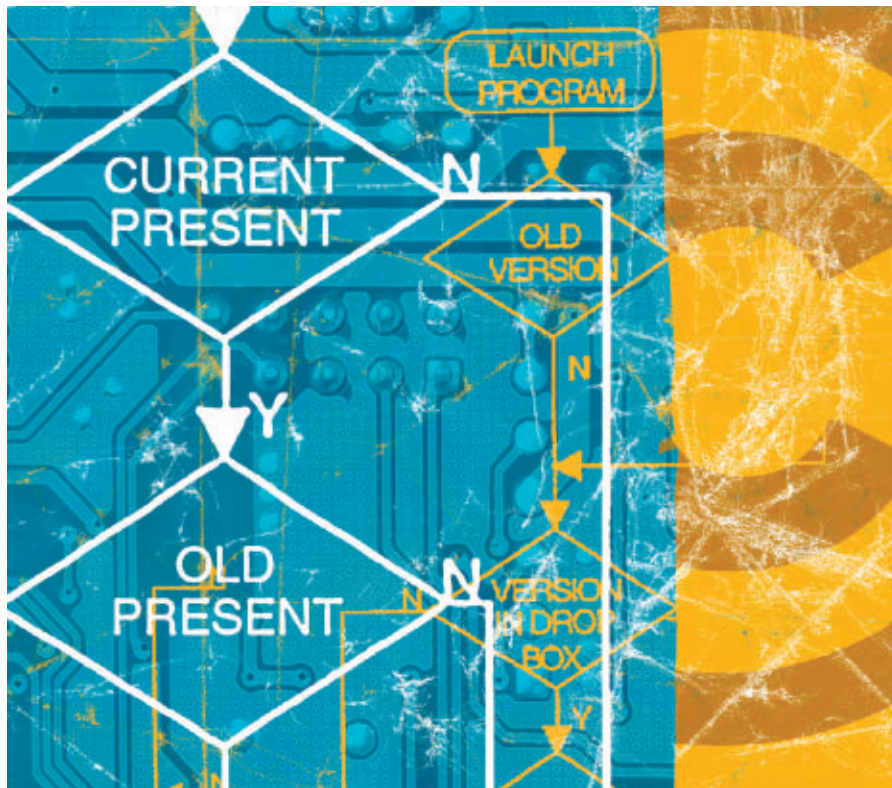
gram to calculate the tensile strength of steel. I want to stress that the other bars to patentability remain. My example of miles-to-kilometers conversion would be patentable subject matter, but, we hope, no patent would be granted because the algorithm is not novel and is obvious. (Though sadly, the PTO seems determined to undermine this hope by granting patents on the most mundane and obvious applications; two excellent books by Besson and Meurer² and by Jaffe and Lerner⁵ explore this point, as well as other deeper problems with the patent system.) But the concern here is not limited to the idea that, without a subject-matter bar, too many obvious patents will be granted

by an overworked and poorly incentivized patent office. It is that the patent was supposed to be granted at the very end of a process of investigation and scientific and engineering innovation. The formulae, algorithms, and scientific discoveries on which the patented invention was based remained in the public domain for all to use. It was only when we got to the very end of the process, with a concrete innovation ready to go to market, that the patent was to be given. Yet the ability to couple the abstract algorithm with the concept of a Turing machine undermines this

ter's slide into debauchery, once that first wall protecting the public domain was breached, the courts found it easier and easier to breach still others. If one could turn an algorithm into a patentable machine (by simply adding "by means of a computer"), then could one not turn a business method into something patentable by specifying the organizational or information technology structure through which the business method is to be implemented?

You might wonder why we would want to patent business methods. Intellectual property rights are supposed to be handed out only when necessary to produce incentives to supply some public good, incentives that otherwise would be lacking. Yet there are already plenty of incentives to come up with new business methods. (Greed and fear are the most obvious.) There is no evidence to believe we need a state-backed monopoly to encourage the development of new business methods. In fact, we want people to copy the businesses of others, lowering the price as a result. The process of copying business methods is called "competition" and is the basis of a free-market economy. Yet patent law would prohibit it for 20 years. So why introduce patents? Brushing aside such minor objections with ease in 1998, in a case called *State Street*, the Court of Appeals for the Federal Circuit declared business methods to be patentable.^d Could this really be what Thomas Jefferson had in mind when he said "I know well the difficulty of drawing a line between the things which are worth to the public the embarrassment of an exclusive patent, and those which are not"?^e I doubt it.

In 2008, the Court of Appeals for the Federal Circuit revisited this ruling in a case called *In re Bilski*.^f Perhaps made wary by several spankings they had recently received at the hands of the U.S.



conception. Suddenly the patents are available at the very beginning of the process, even to people who are merely specifying, in the abstract, the idea of a computer running a particular series of algorithmic activities.

The words "by means of a computer" seem to be an incantation of magical power, able to transubstantiate the ideas and formulae of the public domain into private property.^c And, like the breaking of a minor taboo that presages a Victorian literary charac-

The words "by means of a computer" seem to be an incantation of magical power, able to transubstantiate the ideas and formulae of the public domain into private property.^c And, like the breaking of a minor taboo that presages a Victorian literary charac-

ter's slide into debauchery, once that first wall protecting the public domain was breached, the courts found it easier and easier to breach still others. If one could turn an algorithm into a patentable machine (by simply adding "by means of a computer"), then could one not turn a business method into something patentable by specifying the organizational or information technology structure through which the business method is to be implemented?

^c See, for example, *In re Alappat*, 33 F.3d 1526 (Fed. Cir. 1994); in light of the other cases discussed here, it is a contested issue which parts of this decision survive today.

^d *State St. Bank & Trust Co. v. Signature Fin. Group, Inc.*, 149 F.3d 1368 (Fed. Cir. 1998).


^e Letter from Thomas Jefferson to Isaac McPherson (August 13, 1813) in *The Writings of Thomas Jefferson*, vol. XIII, A.E. Bergh, ed. The Thomas Jefferson Memorial Association of the United States, Washington, D.C., 1907, 326–338; see p. 335 at http://memory.loc.gov/ammem/collections/jefferson_papers/mtjser1.html and follow the "May 1, 1812" hyperlink, then navigate to image 1057.

^f *In re Bernard L. Bilski*, 545 F.3d 943 (Fed. Cir. 2008).


Supreme Court for “creatively interpreting” prior Supreme Court precedent, a majority of the Court of Appeals overturned a portion of the State Street decision. They declared that, to be patentable, an algorithm or method must result in some transformation or be embodied in some machine, rejecting State Street’s more forgiving language, which looked only for some “useful, concrete and tangible result.”^g Patent lawyers too, it seems, have their own metaphysical debates.

But what is the result of all this abstraction? Are business methods patentable? Can an algorithm implemented by a Turing machine thereby be patented? To see how differently *Bilski* could be viewed, one need only compare two of the dissents. Judge Newman lamented the court’s action in restricting patentability and undermining the provision of incentives to meet “the infinite needs of the future”: “It is antithetical to this incentive to restrict eligibility for patenting to what has been done in the past, and foreclose what might be done in the future.”^h Reading the opinion one could almost forget that people have been coming up with business methods all over the world for thousands of years without patent protection, or that having too many patents can be just as harmful to innovation as having too few. Judge Mayer strongly disagreed. “Patenting business methods allows private parties to claim exclusive ownership of ideas and practices which rightfully belong in the public domain.... The patent system is intended to protect and promote advances in science and technology, not ideas about how to structure commercial transactions.”ⁱ In his view, the *Bilski* court was too tame. They had not flatly declared business methods unpatentable, merely changed the metaphysical terms in which those patents needed to be couched. The Supreme Court has granted *certiorari*, meaning that it will hear an appeal of the decision some time in the next year.

The *Bilski* case highlights a larger point. It is commonplace for courts to



The words “by means of a computer” seem to be an incantation of magical power, able to transubstantiate the ideas and formulae of the public domain into private property.



look at the purpose of the law they are enforcing when seeking to understand what it means. In areas of regulation that are obviously “instrumental”—aimed at producing some particular result in the world—this approach is ubiquitous. In applying the antitrust laws, for example, courts have given meaning to the relatively vague words of the law by turning to economic analysis of the likely effects of different rules on different market structures.

Patent law is as instrumental a structure as one could imagine. In the U.S., for example, the constitutional authorization to Congress to make patent and copyright legislation is very explicit that these rights are to be made with a purpose in view. Congress has the power “to promote the progress of science and useful arts, by securing for limited times to authors and inventors the exclusive right to their respective writings and discoveries.”^j One might imagine that courts would try to interpret the patent and copyright laws with that purpose firmly in mind. Yet utilitarian caution about extending monopolies is seldom found in the reasoning of the U.S.’s chief patent court. Until *Bilski*, the court had preferred to quote a phrase from a congressional report that patentable subject matter includes “anything under the sun that is made by man.”^k

The difference is striking. Jefferson said that the job of those who administered the patent system was to see if a patent was worth the embarrassment to the public before granting it. The Constitution tells Congress to make only those patent laws that “promote the progress of science and useful arts.” One might imagine that this constitutional goal would guide courts in construing the same laws. Yet in our chief patent court for the past 20 years, neither Jeffersonian ideals nor the constitutional text has seemed relevant to its thinking when interpreting statutory subject matter. Anything under the sun made by man is patentable subject matter, and there’s an end to it. The case that announced the rule on business methods involved a patent over

g *State Street*, 149 F.3d at 1373.

h *In re Bilski*, 545 F.3d at 998 (Newman, J., dissenting).

i *In re Bilski*, 545 F.3d at 998, 1007 (Mayer, J., dissenting).

j U.S. Constitution, art. I, § 8, cl. 8.

k S. Rep. No. 1979, 82d Cong., 2d Sess., 5 (1952); H.R. Rep. No. 1979, 82d Cong., 2d Sess., 6 (1952).

the process of keeping accounts in a “hub-and-spoke” mutual fund, including multiplying all of the stock holdings of each fund in a family of funds by the respective current share price to get total fund value, then dividing by the number of mutual-fund shares that each customer actually holds to find the balance in their accounts.¹ As my son observed, “I couldn’t do that until nearly the end of third grade!”

In theory of course, if the patent is not novel or obvious, it will still be refused. The Supreme Court recently held that the Court of Appeals for the Federal Circuit made “non-obvious” too easy a standard to meet.² It is unclear, however, whether this judgment will produce concrete effects on actual practices of patent grants and litigation. The PTO’s system puts pressure on examiners to issue patents, and it is very expensive to challenge those that are granted. Better would be, where possible, to rule out certain subject matter (such as business methods) in the first place and more narrowly craft software patents so as to avoid the dangers the copyright decisions anticipated so clearly. Judge Mayer is right. Tempted in part by the power of the metaphor of “idea made machine” in the context of a computer, the Court of Appeals for the Federal Circuit has not been able to bring itself to do so. Where copyright law evolved to wall off, encyst, and minimize the dangers of extending protection over software, patent law initially extended the idea behind software patents to make patentable any thought process that might produce a useful result. Even when it got rid of the “useful result” language, the court was unable to bring itself to declare business methods unpatentable. Once breached, the walls protecting the public domain in patent law show a disturbing tendency to erode at an increasing rate.

To sum up, the conceptual possibilities presented to copyright and patent law by software were fascinating. Should we extend copyright or patent to cover the new technology? The answer was “We will extend both!” Yet the results of the extension were complex and unexpected in ways we should try

to understand if we want to predict the effect of intellectual property on future technologies. Who would have predicted that software copyrights could be used to create a self-perpetuating commons, as well as a monopoly over operating systems, or that judges would talk knowingly of network effects in curtailing the scope of coverage? Who would have predicted that patents would be extended not only to basic algorithms implemented by a computer but to methods of business themselves? (Truly, a strange return to legalized business monopolies for a country whose founders viewed them as one of the greatest evils that could be borne.) The rest of the world has (wisely) been resistant to granting patents over business methods, and even to so-called “pure” software patents. (The empirical evidence, of which there is far too little, suggests that expansive software patents may actually have a negative effect on research and development.¹) Yet as global legal harmonization sweeps onward, little attention is being paid to empirical evidence, and it is not clear which way the norms will tip. Our attitude should be to demand rigorous empirical and economic study before we create or extend legal monopolies. Expansive new rights over emerging technologies may be necessary to encourage innovation, but the case must be made on facts, not faith.

What can we learn from this history? First, we should realize that the mere decision to include a technology within a property regime is only the first in a sequence. As the copyright system showed with software, it is possible to trim protection so as to minimize overreaching. As the business-method patent decisions show us, we don’t always do it. Second, we should understand that we have some new methods of combining property rights and an open “commons” of raw material. The experience of free and open source software should be studied to see whether it has implications for new technologies. We need all the innovation tools we can get. Third, we should be mindful of the fact that much depends on the moment in the development of a technology when property rights begin to be rigorously applied. For better or for worse, property rights came fully to software at a point when no one would have thought

of claiming the most fundamental building blocks—patenting the idea of a Turing machine or the precepts of Boolean algebra. The basics of the field were there for all to build upon. Will that be true with future technologies?

It is disquieting to realize that today the answer to this question is very difficult to provide. In one particular area, synthetic biology, which shares aspects of both software (programming in genetic code) and genetic engineering, there is considerable reason for alarm. As my colleague Arti Rai, and I note in an article on the subject,⁷ it is quite possible to imagine a perfect storm in which the expansive patent law decisions of the past 20 years do to synthetic biology what they could not do to software—lock up the basic building blocks before the field can develop.

The fundamental ideas behind our intellectual property system are sound. Intellectual property rights can be important, even vital, for the development of a particular area of technology. But it is just as easy to harm innovation with rights that are too strong as too weak. The example of software could teach us a lot about the future of good intellectual property policy in high technology, but first we need to pay attention to it. **□**

References

1. Bessen, J. and Hunt, R.M. An empirical look at software patents. *Journal of Economics and Management Strategy* 16, no. 1 (Spring 2007), 157–189.
2. Bessen, J. and Meurer, M.J. *Patent Failure: How Judges, Bureaucrats, and Lawyers Put Innovators at Risk*. Princeton University Press, Princeton, NJ, 2008.
3. Boyle, J. *The Public Domain: Enclosing the Commons of the Mind*. Yale University Press, New Haven, CT, 2008, 185–194.
4. Boyle, J. *Shamans, Software, and Spleens: Law and the Construction of the Information Society*. Harvard University Press, Cambridge, MA, 1996.
5. Jaffe, A. and Lerner, J. *Innovation and Its Discontents: How Our Broken Patent System Is Endangering Innovation and Progress, and What To Do About It*. Princeton University Press, Princeton, NJ, 2004.
6. Samuelson, P., Davis, R., Kapor, M.D., and Reichman, J.H. A manifesto concerning the legal protection of computer programs. *Columbia Law Review* 94, no. 8 (December 1994), 2308–2431.
7. Rai, A. and Boyle, J. Synthetic biology: Caught between property rights, the public domain, and the commons. *PLoS Biology* 5, no. 3 (March 2007): 389–393; <http://www.plosbiology.org/article/info%3Adoi%2F10.1371%2Fjournal.pbio.0050058>.

James Boyle (boyle@law.duke.edu) is William Neal Reynolds Professor of law at Duke Law School, Durham, NC. This article is adapted from his book *The Public Domain: Enclosing the Commons of the Mind* (2008 by Yale University Press), which is freely downloadable from <http://thepublicdomain.org> under a Creative Commons license.

Copyright held by author.

¹ *State Street*, 149 F.3d at 1373.
² *KSR Int’l Co. v. Teleflex Inc.*, 550 U.S. 398 (2007).



Association for
Computing Machinery

Advancing Computing as a Science & Profession



You've come a long way. Share what you've learned.



ACM has partnered with MentorNet, the award-winning nonprofit e-mentoring network in engineering, science and mathematics. MentorNet's award-winning **One-on-One Mentoring Programs** pair ACM student members with mentors from industry, government, higher education, and other sectors.

- Communicate by email about career goals, course work, and many other topics.
- Spend just **20 minutes a week** - and make a huge difference in a student's life.
- Take part in a lively online community of professionals and students all over the world.



Make a difference to a student in your field.

Sign up today at: www.mentornet.net

Find out more at: www.acm.org/mentornet

MentorNet's sponsors include 3M Foundation, ACM, Alcoa Foundation, Agilent Technologies, Amylin Pharmaceuticals, Bechtel Group Foundation, Cisco Systems, Hewlett-Packard Company, IBM Corporation, Intel Foundation, Lockheed Martin Space Systems, National Science Foundation, Naval Research Laboratory, NVIDIA, Sandia National Laboratories, Schlumberger, S.D. Bechtel, Jr. Foundation, Texas Instruments, and The Henry Luce Foundation.

It's one of the fundamental mathematical problems of our time, and its importance grows with the rise of powerful computers.

BY LANCE FORTNOW

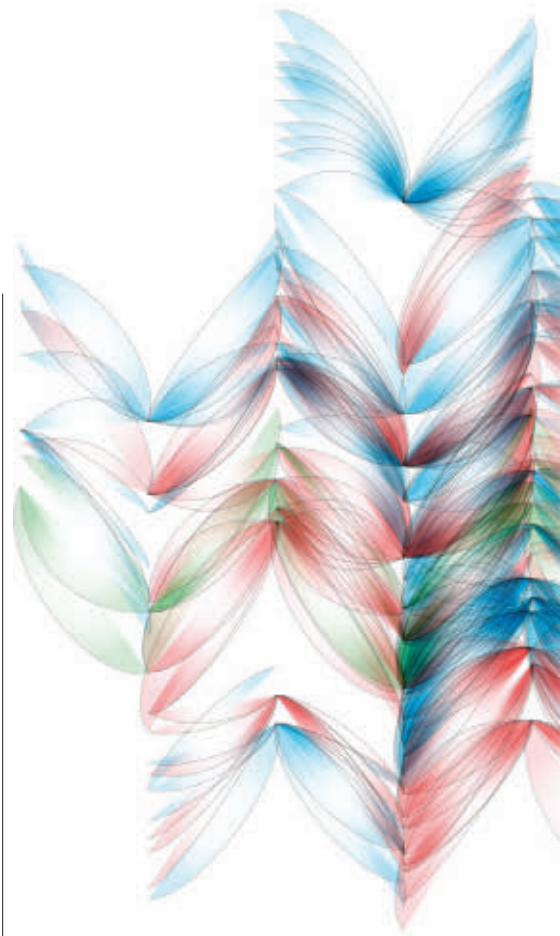
The Status of the P versus NP Problem

WHEN EDITOR-IN-CHIEF MOSHE Vardi asked me to write this piece for *Communications*, my first reaction was the article could be written in two words:

Still open.

When I started graduate school in the mid-1980s, many believed that the quickly developing area of circuit complexity would soon settle the P versus NP problem, whether every algorithmic problem with efficiently verifiable solutions have efficiently computable solutions. But circuit complexity and other approaches to the problem have stalled and we have little reason to believe we will see a proof separating P from NP in the near future.

Nevertheless, the computer science landscape has dramatically changed in the nearly four decades since Steve Cook presented his seminal NP-completeness paper “The Complexity of Theorem-Proving Procedures”¹⁰ in Shaker Heights, OH in early May, 1971. Computational power has dramatically

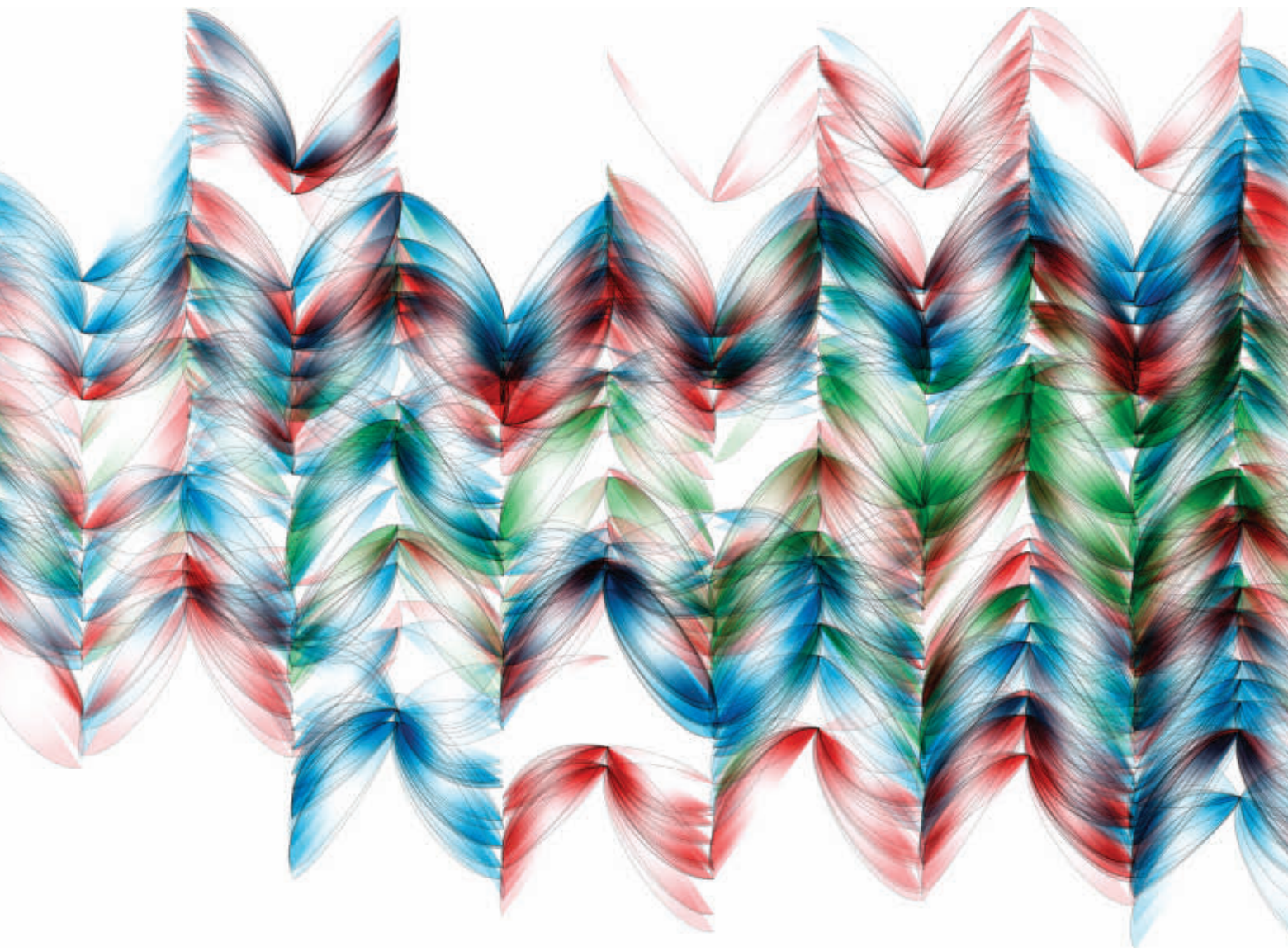


The software written for this illustration makes a stylized version of a network graph that draws connections between elements based on proximity. The graph constantly changes as the elements sort themselves.

increased, the cost of computing has dramatically decreased, not to mention the power of the Internet. Computation has become a standard tool in just about every academic field. Whole subfields of biology, chemistry, physics, economics and others are devoted to large-scale computational modeling, simulations, and problem solving.

As we solve larger and more complex problems with greater computational power and cleverer algorithms, the problems we cannot tackle begin to stand out. The theory of NP-completeness helps us understand these limitations and the P versus NP problem begins to loom large not just as an interesting theoretical question in computer science, but as a basic principle that permeates all the sciences.

So while we don't expect the P versus NP problem to be resolved in the near future, the question has driven research in a number of topics to help us understand, handle, and even take



advantage of the hardness of various computational problems.

In this article I look at how people have tried to solve the **P** versus **NP** problem as well as how this question has shaped so much of the research in computer science and beyond. I will look at how to handle **NP**-complete problems and the theory that has developed from those approaches. I show how a new type of “interactive proof systems” led to limitations of approximation algorithms and consider whether quantum computing can solve **NP**-complete problems (short answer: not likely). And I close by describing a new long-term project that will try to separate **P** from **NP** using algebraic-geometric techniques.

This article does not try to be totally accurate or complete either technically or historically, but rather informally describes the **P** versus **NP** problem and the major directions in computer science inspired by this question over the past several decades.

What is the **P** versus **NP** Problem?

Suppose we have a large group of students that we need to pair up to work on projects. We know which students are compatible with each other and we want to put them in compatible groups of two. We could search all possible pairings but even for 40 students we would have more than 300 billion trillion possible pairings.

In 1965, Jack Edmonds¹² gave an efficient algorithm to solve this matching problem and suggested a formal definition of “efficient computation” (runs in time a fixed polynomial of the input size). The class of problems with efficient solutions would later become known as **P** for “Polynomial Time.”

But many related problems do not seem to have such an efficient algorithm. What if we wanted to make groups of three students with each pair of students in each group compatible (Partition into Triangles)? What if we wanted to find a large group of students all of whom are compatible with each

other (Clique)? What if we wanted to sit students around a large round table with no incompatible students sitting next to each other (Hamiltonian Cycle)? What if we put the students into three groups so that each student is in the same group with only his or her compatibles (3-Coloring)?

All these problems have a similar favor: Given a potential solution, for example, a seating chart for the round table, we can validate that solution efficiently. The collection of problems that have efficiently verifiable solutions is known as **NP** (for “Nondeterministic Polynomial-Time,” if you have to ask).

So **P** = **NP** means that for every problem that has an efficiently verifiable solution, we can find that solution efficiently as well.

We call the very hardest **NP** problems (which include Partition into Triangles, Clique, Hamiltonian Cycle and 3-Coloring) “**NP**-complete,” that is, given an efficient algorithm for one of them, we can find an efficient algorithm for all

of them and in fact any problem in **NP**. Steve Cook, Leonid Levin, and Richard Karp^{10, 24, 27} developed the initial theory of **NP**-completeness that generated multiple ACM Turing Awards.

In the 1970s, theoretical computer scientists showed hundreds more problems **NP**-complete (see Garey and Johnson¹⁶). An efficient solution to any **NP**-complete problem would imply $\mathbf{P} = \mathbf{NP}$ and an efficient solution to every **NP**-complete problem.

Most computer scientists quickly came to believe $\mathbf{P} \neq \mathbf{NP}$ and trying to prove it quickly became the single most important question in all of theoretical computer science and one of the most important in all of mathematics. Soon the **P** versus **NP** problem became an important computational issue in nearly every scientific discipline.

As computers grew cheaper and more powerful, computation started playing a major role in nearly every academic field, especially the sciences. The more scientists can do with computers, the more they realize some problems seem computationally difficult. Many of these fundamental problems turn out to be **NP**-complete. A small sample:

- ▶ Finding a DNA sequence that best fits a collection of fragments of the sequence (see Gusfield²⁰).
- ▶ Finding a ground state in the Ising model of phase transitions (see Cibra⁸).
- ▶ Finding Nash Equilibriums with specific properties in a number of environments (see Conitzer⁹).
- ▶ Finding optimal protein threading procedures.²⁶
- ▶ Determining if a mathematical statement has a short proof (follows from Cook¹⁰).

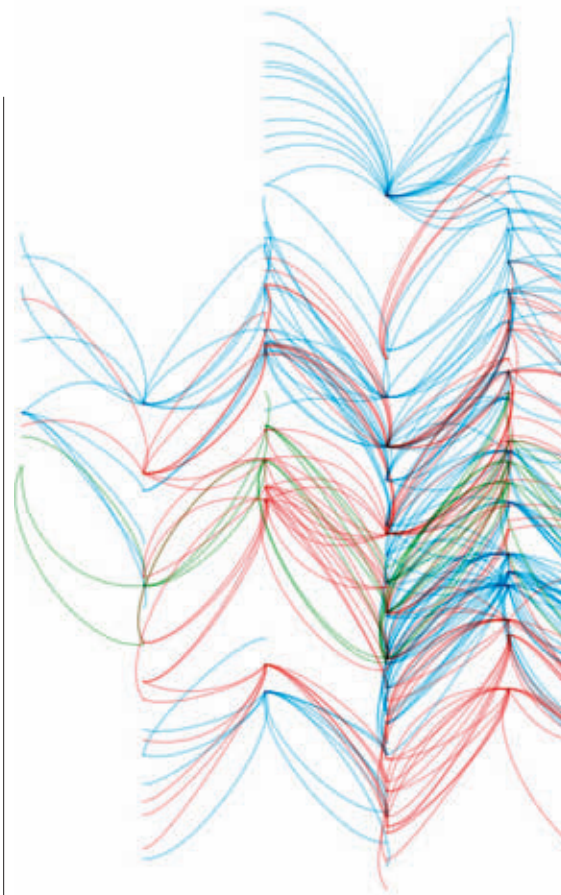
In 2000, the Clay Math Institute named the **P** versus **NP** problem as one of the seven most important open questions in mathematics and has offered a million-dollar prize for a proof that determines whether or not $\mathbf{P} = \mathbf{NP}$.

What If $\mathbf{P} = \mathbf{NP}$?

To understand the importance of the **P** versus **NP** problem let us imagine a world where $\mathbf{P} = \mathbf{NP}$. Technically we could have $\mathbf{P} = \mathbf{NP}$, but not have practical algorithms for most **NP**-complete problems. But suppose in fact we do have very quick algorithms for all these problems.

Many focus on the negative, that if

What we would gain from $\mathbf{P} = \mathbf{NP}$ will make the whole Internet look like a footnote in history.

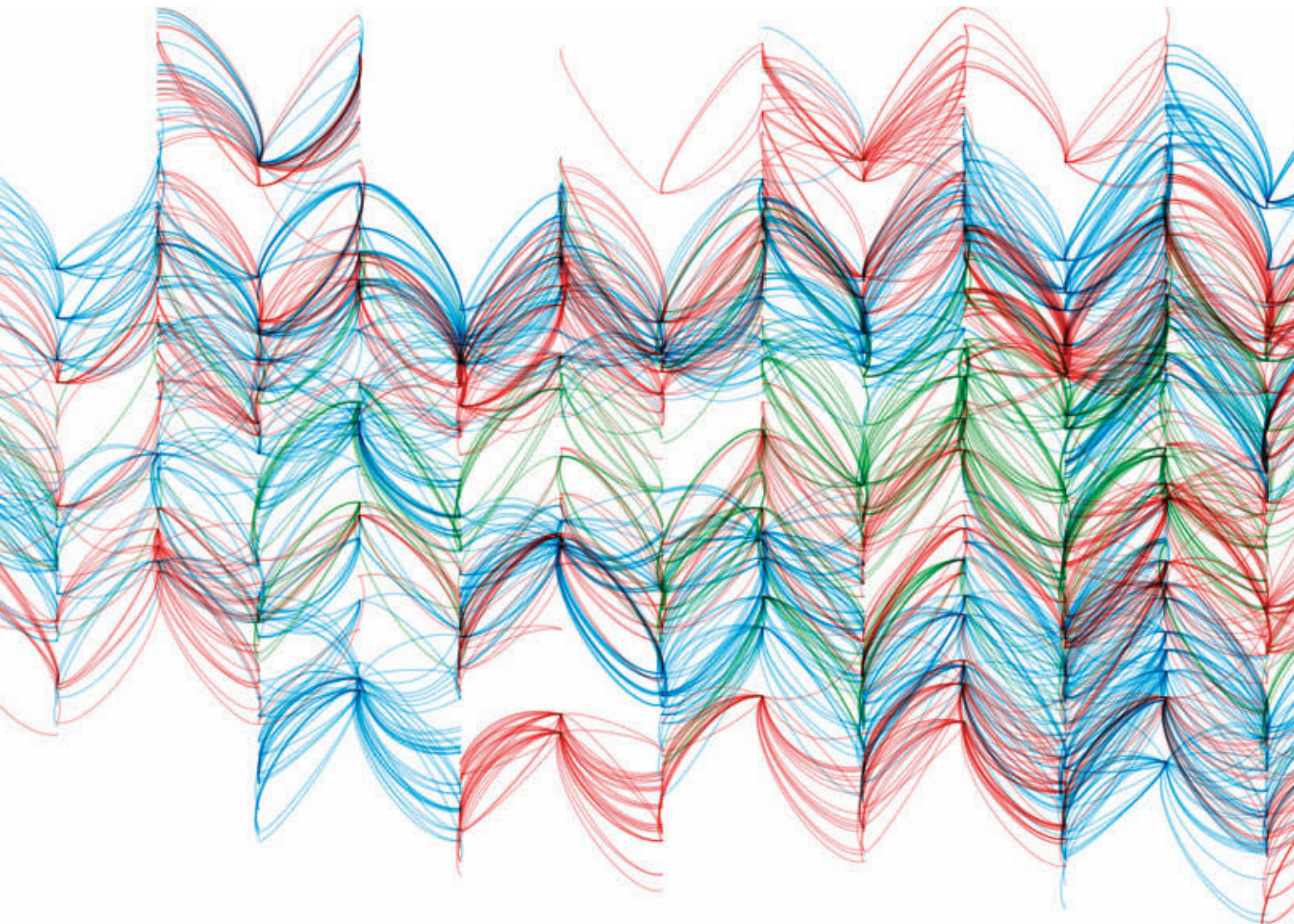


$\mathbf{P} = \mathbf{NP}$ then public-key cryptography becomes impossible. True, but what we will gain from $\mathbf{P} = \mathbf{NP}$ will make the whole Internet look like a footnote in history.

Since all the **NP**-complete optimization problems become easy, everything will be much more efficient. Transportation of all forms will be scheduled optimally to move people and goods around quicker and cheaper. Manufacturers can improve their production to increase speed and create less waste. And I'm just scratching the surface.

Learning becomes easy by using the principle of Occam's razor—we simply find the smallest program consistent with the data. Near perfect vision recognition, language comprehension and translation and all other learning tasks become trivial. We will also have much better predictions of weather and earthquakes and other natural phenomenon.

$\mathbf{P} = \mathbf{NP}$ would also have big implications in mathematics. One could find short, fully logical proofs for theorems



but these proofs are usually extremely long. But we can use the Occam razor principle to recognize and verify mathematical proofs as typically written in journals. We can then find proofs of theorems that have reasonable length proofs say in under 100 pages. A person who proves $P = NP$ would walk home from the Clay Institute not with \$1 million check but with seven (actually six since the Poincaré Conjecture appears solved).

Don't get your hopes up. Complexity theorists generally believe $P \neq NP$ and such a beautiful world cannot exist.

Approaches to Showing $P \neq NP$

Here, I present a number of ways we have tried and failed to prove $P \neq NP$. The survey of Fortnow and Homer¹⁴ gives a fuller historical overview of these techniques.

Diagonalization. Can we just construct an NP language L specifically designed so that every single polynomial-time algorithm fails to compute L properly on some input? This approach,

known as diagonalization, goes back to the 19th century.

In 1874, Georg Cantor⁷ showed the real numbers are uncountable using a technique known as diagonalization. Given a countable list of reals, Cantor showed how to create a new real number not on that list.

Alan Turing, in his seminal paper on computation,³⁸ used a similar technique to show that the Halting problem is not computable. In the 1960s complexity theorists used diagonalization to show that given more time or memory one can solve more problems. Why not use diagonalization to separate NP from P ?

Diagonalization requires simulation and we don't know how a fixed NP machine can simulate an arbitrary P machine. Also a diagonalization proof would likely relativize, that is, work even if all machines involved have access to the same additional information. Baker, Gill and Solovay⁶ showed no relativizable proof can settle the P versus NP problem in either direction.

Complexity theorists have used di-

agonalization techniques to show some NP -complete problems like Boolean formula satisfiability cannot have algorithms that use both a small amount of time and memory,³⁹ but this is a long way from $P \neq NP$.

Circuit Complexity. To show $P \neq NP$ it is sufficient to show some NP -complete problem cannot be solved by relatively small circuits of AND, OR, and NOT gates (the number of gates bounded by a fixed polynomial in the input size).

In 1984, Furst, Saxe, and Sipser¹⁵ showed that small circuits cannot solve the parity function if the circuits have a fixed number of layers of gates. In 1985, Razborov³¹ showed the NP -complete problem of finding a large clique does not have small circuits if one only allows AND and OR gates (no NOT gates). If one extends Razborov's result to general circuits one will have proved $P \neq NP$.

Razborov later showed his techniques would fail miserably if one allows NOT gates.³² Razborov and Rudich³³ develop a notion of "natural" proofs and give evidence that our limited techniques

in circuit complexity cannot be pushed much further. And, in fact, we haven't seen any significantly new circuit lower bounds in the past 20 years.

Proof Complexity. Consider the set of Tautologies, the Boolean formulas \emptyset of variables over ANDs, ORs, and NOTs such that every setting of the variables to True and False makes \emptyset true, for example the formula

$$(x \text{ AND } y) \text{ OR } (\text{NOT } x) \text{ OR } (\text{NOT } y).$$

A literal is a variable or its negation, such as x or $\text{NOT } x$. A formula, like the one here, is in Disjunctive Normal Form (DNF) if it is the OR of ANDs of one or more literals.

If a formula \emptyset is not a tautology, we can give an easy proof of that fact by exhibiting an assignment of the variables that makes \emptyset false. But if \emptyset were indeed a tautology, we don't expect short proofs. If one could prove there are no short proofs of tautology that would imply $\mathbf{P} \neq \mathbf{NP}$.

Resolution is a standard approach to proving tautologies of DNFs by finding two clauses of the form $(\psi_1 \text{ AND } x)$ and $(\psi_2 \text{ AND NOT } x)$ and adding the clause $(\psi_1 \text{ AND } \psi_2)$. A formula is a tautology exactly when one can produce an empty clause in this manner.

In 1985, Wolfgang Haken²¹ showed that tautologies that encode the pigeon-hole principle ($n + 1$ pigeons in n holes means some hole has more than one pigeon) do not have short resolution proofs.

Since then complexity theorists have shown similar weaknesses in a number of other proof systems including cutting planes, algebraic proof systems based on polynomials, and restricted versions of proofs using the Frege axioms, the basic axioms one learns in an introductory logic course.

But to prove $\mathbf{P} \neq \mathbf{NP}$ we would need to show that tautologies cannot have short proofs in an arbitrary proof system. Even a breakthrough result showing tautologies don't have short general Frege proofs would not suffice in separating \mathbf{NP} from \mathbf{P} .

Dealing with Hardness

So you have an \mathbf{NP} -complete problem you just have to solve. If, as we believe, $\mathbf{P} \neq \mathbf{NP}$ you won't find a general algorithm that will correctly and accurately solve

your problem all the time. But sometimes you need to solve the problem anyway. All hope is not lost. Here, I describe some of the tools one can use on \mathbf{NP} -complete problems and how computational complexity theory studies these approaches. Typically one needs to combine several of these approaches when tackling \mathbf{NP} -complete problems in the real world.

Brute Force. Computers have gotten faster, much faster since \mathbf{NP} -completeness was first developed. Brute force search through all possibilities is now possible for some small problem instances. With some clever algorithms we can even solve some moderate size problems with ease.

The \mathbf{NP} -complete traveling salesperson problem asks for the smallest distance tour through a set of specified cities. Using extensions of the cutting-plane method we can now solve, in practice, traveling salespeople problems with more than 10,000 cities (see Applegate³).

Consider the 3SAT problem, solving Boolean formula satisfiability where formulas are in the form of the AND of several clauses where each clause is the OR of three literal variables or negations of variables). 3SAT remains \mathbf{NP} -complete but the best algorithms can in practice solve SAT problems on about 100 variables. We have similar results for other variations of satisfiability and many other \mathbf{NP} -complete problems.

But for satisfiability on general formulae and on many other \mathbf{NP} -complete problems we do not know algorithms better than essentially searching all the possibilities. In addition, all these algorithms have exponential growth in their running times, so even a small increase in the problem size can kill what was an efficient algorithm. Brute force alone will not solve \mathbf{NP} -complete problems no matter how clever we are.

Parameterized Complexity. Consider the Vertex Cover problem, find a set of k "central people" such that for every compatible pair of people, at least one of them is central. For small k we can determine whether a central set of people exists efficiently no matter the total number n of people we are considering. For the Clique problem even for small k the problem can still be difficult.

Downey and Fellows¹¹ developed a theory of parameterized complexity that

gives a fine-grained analysis of the complexity of \mathbf{NP} -complete problems based on their parameter size.

Approximation. We cannot hope to solve \mathbf{NP} -complete optimization problems exactly but often we can get a good approximate answer. Consider the traveling salesperson problem again with distances between cities given as the crow flies (Euclidean distance). This problem remains \mathbf{NP} -complete but Aro⁴ gives an efficient algorithm that gets very close to the best possible route.

Consider the MAX-CUT problem of dividing people into two groups to maximize the number of incompatibles between the groups. Goemans and Williamson¹⁷ use semi-definite programming to give a division of people only a .878567 factor of the best possible.

Heuristics and Average-Case Complexity. The study of \mathbf{NP} -completeness focuses on how algorithms perform on the worst possible inputs. However the specific problems that arise in practice may be much easier to solve. Many computer scientists employ various heuristics to solve \mathbf{NP} -complete problems that arise from the specific problems in their fields.

While we create heuristics for many of the \mathbf{NP} -complete problems, Boolean formula Satisfiability (SAT) receives more attention than any other. Boolean formulas, especially those in conjunctive normal form (CNF), the AND of ORs of variables and their negations, have a very simple description and yet are general enough to apply to a large number of practical scenarios particularly in software verification and artificial intelligence. Most natural \mathbf{NP} -complete problems have simple efficient reductions to the satisfiability of Boolean formulas. In competition these SAT solvers can often settle satisfiability of formulas of one million variables.^a

Computational complexity theorists study heuristics by considering average-case complexity—how well can algorithms perform on average from instances generated by some specific distribution.

Leonid Levin²⁸ developed a theory of efficient algorithms over a specific distribution and formulated a distributional version of the \mathbf{P} versus \mathbf{NP} problem.

Some problems, like versions of the

a <http://www.satcompetition.org>.

shortest vector problem in a lattice or computing the permanent of a matrix, are hard on average exactly when they are hard on worst-case inputs, but neither of these problems is believed to be **NP**-complete. Whether similar worst-to-average reductions hold for **NP**-complete sets is an important open problem.

Average-case complexity plays an important role in many areas of computer science, particularly cryptography, as discussed later.

Interactive Proofs and Limits of Approximation

Previously, we saw how sometimes one can get good approximate solutions to **NP**-complete optimization problems. Many times though we seem to hit a limit on our ability to even get good approximations. We now know that we cannot achieve better approximations on many of these problems unless $\mathbf{P} = \mathbf{NP}$ and we could solve these problems exactly. The techniques to show these negative results came out of a new model of proof system originally developed for cryptography and to classify group theoretic algorithmic problems.

As mentioned earlier, we don't expect to have short traditional proofs of tautologies. But consider an "interactive proof" model where a prover Peggy tries to convince a verifier Victor that a formula ϕ is a tautology. Victor can ask Peggy randomly generated questions and need only be convinced with high confidence. Quite surprisingly, these proof systems have been shown to exist not only for tautologies but for any problem computable in a reasonable amount of memory.

A variation known as a "probabilistically checkable proof system" (PCPs), where Peggy writes down an encoded proof and Victor can make randomized queries to the bits of the proof, has applications for approximations. The "PCP Theorem" optimizes parameters, which in its strong form shows that every language in **NP** has a PCP where Victor uses a tiny number of random coins and queries only three bits of the proof.

One can use this PCP theorem to show the limitations of approximation for a large number of optimization questions. For example, one cannot approximate the largest clique in a group of n people by more than a multiplicative ra-



We expect $\mathbf{P} \neq \mathbf{NP}$ to hold in very strong ways. We can use strong hardness assumptions as a positive tool, particularly to create cryptographic protocols and to reduce or even eliminate the need of random bits in probabilistic algorithms.



tio of nearly \sqrt{n} unless $\mathbf{P} = \mathbf{NP}$. See Madhu Sudan's recent article in *Communications* for more details and references on PCPs.³⁶

One can do even better assuming a "Unique Games Conjecture" that there exists PCPs for **NP** problems with some stronger properties. Consider the MAX-CUT problem of dividing people discussed earlier. If the unique games conjecture holds one cannot do better than the .878567 factor given by the Goemans-Williamson approximation algorithm.²⁶ Recent work shows how to get a provably best approximation for essentially any constrained problem assuming this conjecture.³⁰

Using Hardness

In "What If $\mathbf{P} = \mathbf{NP}$?" we saw the nice world that arises when we assume $\mathbf{P} = \mathbf{NP}$. But we expect $\mathbf{P} \neq \mathbf{NP}$ to hold in very strong ways. We can use strong hardness assumptions as a positive tool, particularly to create cryptographic protocols and to reduce or even eliminate the need of random bits in probabilistic algorithms.

Cryptography. We take it for granted these days, the little key or lock on our Web page that tells us that someone listening to the network won't get the credit card number I just sent to an online store or the password to the bank that controls my money. But public-key cryptography, the ability to send secure messages between two parties that have never privately exchanged keys, is a relatively new development based on hardness assumptions of computational problems.

If $\mathbf{P} = \mathbf{NP}$ then public-key cryptography is impossible. Assuming $\mathbf{P} \neq \mathbf{NP}$ is not enough to get public-key protocols, instead we need strong average-case assumptions about the difficulty of factoring or related problems.

We can do much more than just public-key cryptography using hard problems. Suppose Alice's husband Bob is working on a Sudoku puzzle and Alice claims she has a solution to the puzzle (solving a $n \times n$ Sudoku puzzle is **NP**-complete). Can Alice convince Bob that she knows a solution without revealing any piece of it?

Alice can use a "zero-knowledge proof," an interactive proof with the additional feature that the verifier learns nothing other than some property

holds, like a Sudoku puzzle having a solution. Every NP search problem has a zero-knowledge proof under the appropriate hardness assumptions.

Online poker is generally played through some “trusted” Web site, usually somewhere in the Caribbean. Can we play poker over the Internet without a trusted server? Using the right cryptographic assumptions, not only poker but any protocol that uses a trusted party can be replaced by one that uses no trusted party and the players can’t cheat or learn anything new beyond what they could do with the trusted party.^b

Eliminating Randomness. In the 1970s we saw a new type of algorithm, one that used random bits to aid in finding a solution to a problem. Most notably we had probabilistic algorithms³⁵ for determining whether a number is prime, an important routine needed for modern cryptography. In 2004, we discovered we don’t need randomness at all to efficiently determine if a number is prime.² Does randomness help us at all in finding solutions to NP problems?

Truly independent and uniform random bits are either very difficult or impossible to produce (depending on your beliefs about quantum mechanics). Computer algorithms instead use pseudorandom generators to generate a sequence of bits from some given seed. The generators typically found on our computers usually work well but occasionally give incorrect results both in theory and in practice.

We can create theoretically better pseudorandom generators in two different ways, one based on the strong hardness assumptions of cryptography and the other based on worst-case complexity assumptions. I will focus on this second approach.

We need to assume a bit more than $P \neq NP$, roughly that NP-complete problems cannot be solved by smaller than expected AND-OR-NOT circuits. A long series of papers showed that, under this assumption, any problem with an efficient probabilistic algorithm also has an efficient algorithm that uses a pseudorandom generator with a very short seed, a surprising connection between hard languages and pseudo-randomness (see Impagliazzo²³). The seed is so short we can try all possible seeds effi-

ciently and avoid the need for randomness altogether.

Thus complexity theorists generally believe having randomness does not help in solving NP search problems and that NP-complete problems do not have efficient solutions, either with or without using truly random bits.

While randomness doesn’t seem necessary for solving search problems, the unpredictability of random bits plays a critical role in cryptography and interactive proof systems and likely cannot be avoided in these scenarios.

Could Quantum Computers Solve NP-Complete Problems?

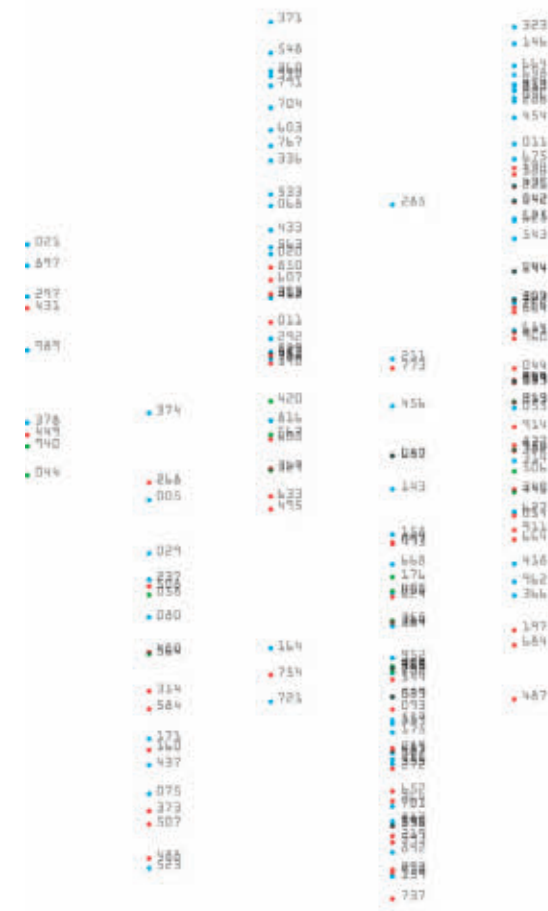
While we have randomized and non-randomized efficient algorithms for determining whether a number is prime, these algorithms usually don’t give us the factors of a composite number. Much of modern cryptography relies on the fact that factoring or similar problems do not have efficient algorithms.

In the mid-1990s, Peter Shor³⁴ showed how to factor numbers using a hypothetical quantum computer. He also developed a similar quantum algorithm to solve the discrete logarithm problem. The hardness of discrete logarithm on classical computers is also used as a basis for many cryptographic protocols. Nevertheless, we don’t expect that factoring or finding discrete logarithms are NP-complete. While we don’t think we have efficient algorithms to solve factoring or discrete logarithm, we also don’t believe we can reduce NP-complete problems like Clique to the factoring or discrete logarithm problems.

So could quantum computers one day solve NP-complete problems? Unlikely.

I’m not a physicist so I won’t address the problem as to whether these machines can actually be built at a large enough scale to solve factoring problems larger than we can with current technology (about 200 digits). After billions of dollars of funding of quantum computing research we still have a long way to go.

Even if we could build these machines, Shor’s algorithm relies heavily on the algebraic structures of numbers that we don’t see in the known NP-complete problems. We know that his algorithm cannot be applied to generic “black-box”



NP can be seen as a graph where every element is connected to every other element. Over these pages a deconstruction of the graph is shown.

search problems so any algorithm would have to use some special structure of NP-complete problems that we don’t know about. We have used some algebraic structure of NP-complete problems for interactive and zero-knowledge proofs but quantum algorithms would seem to require much more.

Lov Grover¹⁹ did find a quantum algorithm that works on general NP problems but that algorithm only achieves a quadratic speed-up and we have evidence that those techniques will not go further.

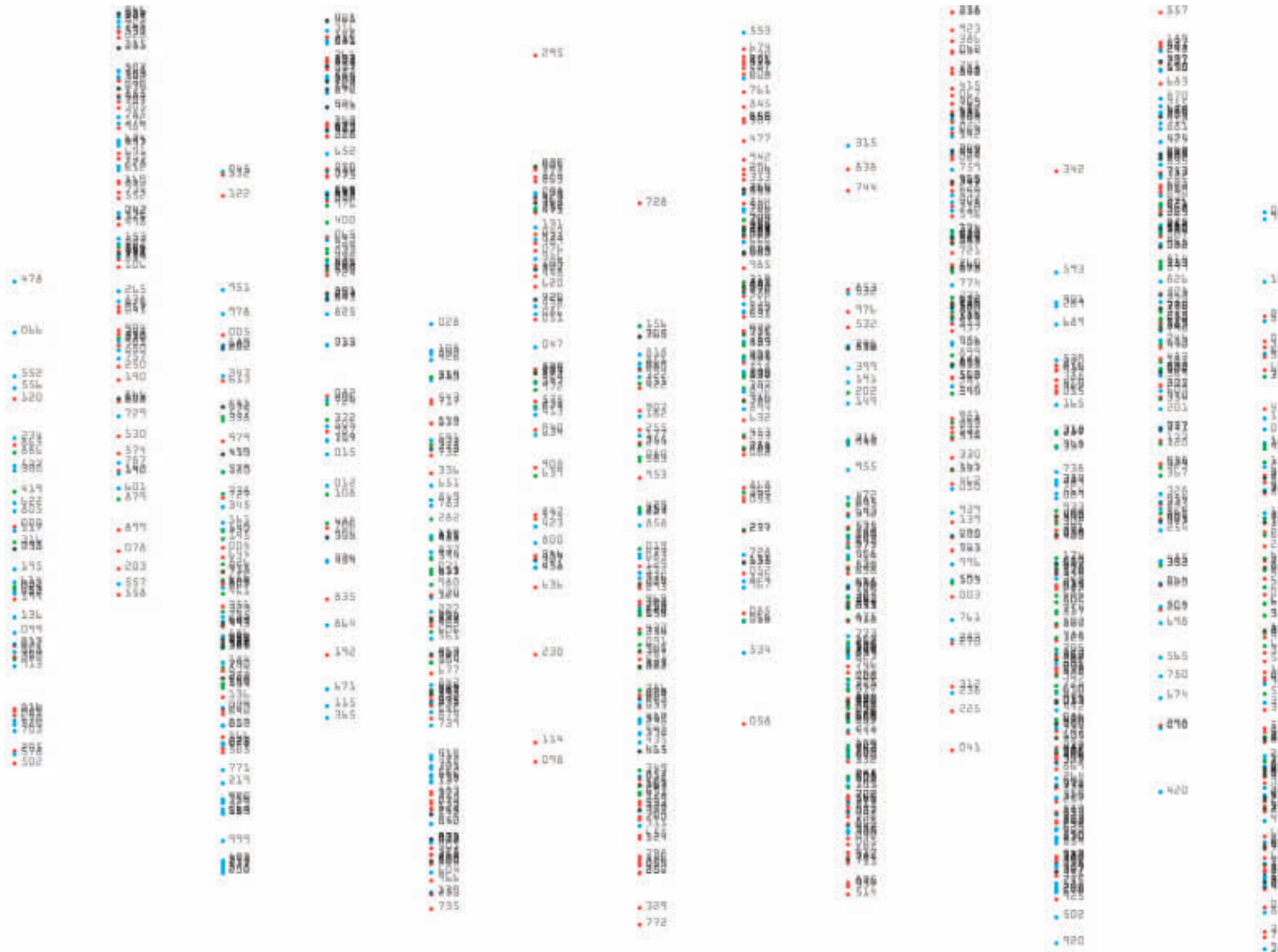
Meanwhile quantum cryptography, using quantum mechanics to achieve some cryptographic protocols without hardness assumptions, has had some success both in theory and in practice.

A New Hope?

Ketan Mulmuley and Milind Sohoni have presented an approach to the P versus NP problem through algebraic geometry, dubbed Geometric Complexity Theory, or GCT.²⁹

This approach seems to avoid the dif-

b See the survey of Goldreich¹⁸ for details.



difficulties mentioned earlier, but requires deep mathematics that could require many years or decades to carry through.

In essence, they define a family of high-dimension polygons P_n based on group representations on certain algebraic varieties. Roughly speaking, for each n , if P_n contains an integral point, then any circuit family for the Hamiltonian path problem must have size at least $n^{\log n}$ on inputs of size n , which implies $\mathbf{P} \neq \mathbf{NP}$. Thus, to show that $\mathbf{P} \neq \mathbf{NP}$ it suffices to show that P_n contains an integral point for all n .

Although all that is necessary is to show that P_n contains an integral point for all n , Mulmuley and Sohoni argue that this direct approach would be difficult and instead suggest first showing that the integer programming problem for the family P_n is, in fact, in \mathbf{P} . Under this approach, there are three significant steps remaining:

1. Prove that the LP relaxation solves the integer programming problem for P_n in polynomial time;
2. Find an efficient, simple combi-

natorial algorithm for the integer programming problem for P_n , and;

3. Prove that this simple algorithm always answers “yes.”

Since the polygons P_n are algebro-geometric in nature, solving (1) is thought to require algebraic geometry, representation theory, and the theory of quantum groups. Mulmuley and Sohoni have given reasonable algebro-geometric conditions that imply (1). These conditions have classical analogues that are known to hold, based on the Riemann Hypothesis over finite fields (a theorem proved by André Weil in the 1960s). Mulmuley and Sohoni suggest that an analogous Riemann Hypothesis-like statement is required here (though not the classical Riemann Hypothesis).

Although step (1) is difficult, Mulmuley and Sohoni have provided definite conjectures based on reasonable mathematical analogies that would solve (1). In contrast, the path to completing steps (2) and (3) is less clear. Despite these remaining hurdles, even solving the conjectures involved in (1) could

provide some insight to the \mathbf{P} versus \mathbf{NP} problem.

Mulmuley and Sohoni have reduced a question about the nonexistence of polynomial-time algorithms for all \mathbf{NP} -complete problems to a question about the existence of a polynomial-time algorithm (with certain properties) for a specific problem. This should give us some hope, even in the face of problems (1)–(3).

Nevertheless, Mulmuley believes it will take about 100 years to carry out this program, if it works at all.

Conclusion

This survey focused on the \mathbf{P} versus \mathbf{NP} problem, its importance, our attempts to prove $\mathbf{P} \neq \mathbf{NP}$ and the approaches we use to deal with the \mathbf{NP} -complete problems that nature and society throws at us. Much of the work mentioned required a long series of mathematically difficult research papers that I could not hope to adequately cover in this short article. Also the field of computational complexity goes well

beyond just the **P** versus **NP** problem that I haven't discussed here. In "Further Reading," a number of references are presented for those interested in a deeper understanding of the **P** versus **NP** problem and computational complexity.

The **P** versus **NP** problem has gone from an interesting problem related to logic to perhaps the most fundamental and important mathematical question of our time, whose importance only grows as computers become more powerful and widespread. The question has even hit popular culture appearing in television shows such as *The Simpsons* and *Numb3rs*. Yet many only know of the basic principles of **P** versus **NP** and I hope this survey has given you a small feeling of the depth of research inspired by this mathematical problem.

Proving $P \neq NP$ would not be the end of the story, it would just show that **NP**-complete problem, don't have efficient algorithms for all inputs but many questions might remain. Cryptography, for example, would require that a problem like factoring (not believed to be **NP**-complete) is hard for randomly drawn composite numbers.

Proving $P \neq NP$ might not be the start of the story either. Weaker separations remain perplexingly difficult, for example showing that Boolean-formula Satisfiability cannot be solved in near-linear time or showing that some problem using a certain amount of memory cannot be solved using roughly the same amount of time.

None of us truly understands the **P** versus **NP** problem, we have only begun to peel the layers around this increasingly complex question. Perhaps we will see a resolution of the **P** versus **NP** problem in the near future but I almost hope not. The **P** versus **NP** problem continues to inspire and boggle the mind and continued exploration of this problem will lead us to yet even new complexities in that truly mysterious process we call computation.

Further Reading

Recommendations for a more in-depth look at the **P** versus **NP** problem and the other topics discussed in this article:

- ▶ Steve Homer and I have written a detailed historical view of computational complexity.¹⁴
- ▶ The 1979 book of Garey and John-

son still gives the best overview of the **P** versus **NP** problem with an incredibly useful list of **NP**-complete problems.¹⁶

- ▶ Scott Aaronson looks at the unlikely possibility that the **P** versus **NP** problem is formally independent.¹

- ▶ Russell Impagliazzo gives a wonderful description of five possible worlds of complexity.²²

- ▶ Sanjeev Arora and Boaz Barak have a new computational complexity textbook with an emphasis on recent research directions.⁵

- ▶ The *Foundations and Trends in Theoretical Computer Science* journal and the Computational Complexity columns of the *Bulletin of the European Association of Theoretical Computer Science* and *SI-GACT News* have many wonderful surveys on various topics in theory including those mentioned in this article.

- ▶ Read the blog Computational Complexity and you will be among the first to know about any updates of the status of the **P** versus **NP** problem.¹³

Acknowledgments

Thanks to Rahul Santhanam for many useful discussions and comments. Josh Grochow wrote an early draft. The anonymous referees provided critical advice. Some of the material in this article has appeared in my earlier surveys and my blog.¹³ C

References

1. Aaronson, S. Is P versus NP formally independent? *Bulletin of the European Association for Theoretical Computer Science* 81 (Oct. 2003).
2. Agrawal, M., Kayal, N., and Saxena, N. PRIMES. In *Annals of Mathematics* 160, 2 (2004) 781–793.
3. Applegate, D., Bixby, R., Chvátal, V., and Cook, W. On the solution of traveling salesman problems. *Documenta Mathematica*, Extra Volume ICM III (1998), 645–656.
4. Arora, S. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *J. ACM* 45, 5 (Sept. 1998), 753–782.
5. Arora, S. and Barak, B. *Complexity Theory: A Modern Approach*. Cambridge University Press, Cambridge, 2009.
6. Baker, T., Gill, J., and Solovay, R. Relativizations of the P = NP question. *SIAM Journal on Computing* 4, 4 (1975), 431–442.
7. Cantor, G. Ueber eine Eigenschaft des Inbegriffes aller reellen algebraischen Zahlen. *Crelle's Journal* 77 (1874), 258–262.
8. Cjpra, B. This Ising model is NP-complete. *SIAM News* 33, 6 (July/Aug. 2000).
9. Conitzer, V. and Sandholm, T. New complexity results about Nash equilibria. *Games and Economic Behavior* 63, 2 (July 2008), 621–641.
10. Cook, S. The complexity of theorem-proving procedures. In *Proceedings of the 3rd ACM Symposium on the Theory of Computing*, ACM, NY, 1971, 151–158.
11. Downey, R. and Fellows, M. *Parameterized Complexity*. Springer, 1999.
12. Edmonds, J. Paths, trees and owers. *Canadian Journal of Mathematics* 17, (1965), 449–467.
13. Fortnow, L. and Gasarch, W. Computational complexity; <http://weblog.fortnow.com>.
14. Fortnow, L. and Homer, S. A short history of computational complexity. *Bulletin of the European Association for Theoretical Computer Science* 80,

- (June 2003).
15. Furst, M., Saxe, J., and Sipser, M. Parity, circuits and the polynomial-time hierarchy. *Mathematical Systems Theory* 17 (1984), 13–27.
16. Garey, M. and Johnson, D. *Computers and Intractability. A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, NY, 1979.
17. Goemans, M. and Williamson, D. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM* 42, 6 (1995), 1115–1145.
18. Goldreich, O. Foundations of cryptographyla primer. *Foundations and Trends in Theoretical Computer Science* 1, 1 (2005) 1–116.
19. Grover, L. A fast quantum mechanical algorithm for database search. In *Proceedings of the 28th ACM Symposium on the Theory of Computing*. ACM, NY, 1996, 212–219.
20. Gusfield, D. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*. Cambridge University Press, 1997.
21. Haken, A. The intractability of resolution. *Theoretical Computer Science*, 39 (1985) 297–305.
22. Impagliazzo, R. A personal view of average-case complexity theory. In *Proceedings of the 10th Annual Conference on Structure in Complexity Theory*. IEEE Computer Society Press, 1995, 134–147.
23. Impagliazzo, R. and Wigderson, A. P = BPP if E requires exponential circuits: Derandomizing the XOR lemma. In *Proceedings of the 29th ACM Symposium on the Theory of Computing*. ACM, NY, 1997, 220–229.
24. Karp, R. Reducibility among combinatorial problems. *Complexity of Computer Computations*. R. Miller and J. Thatcher, Eds. Plenum Press, 1972, 85–103.
25. Khot, S., Kindler, G., Mossel, E., and O'Donnell, R. Optimal inapproximability results for MAX-CUT and other 2-variable CSPs? *SIAM Journal on Computing* 37, 1 (2007), 319–357.
26. Lathrop, R. The protein threading problem with sequence amino acid interaction preferences is NP-complete. *Protein Engineering* 7, 9 (1994), 1059–1068.
27. Levin, L. Universal'nyie perebornnye zadachi (Universal search problems: in Russian). *Problemy Peredachi Informatsii* 9, 3 (1973), 265–266. Corrected English translation.³⁷
28. Levin, L. Average case complete problems. *SIAM Journal on Computing* 15, (1986), 285–286.
29. Mulmuley, K. and Sohoni, M. Geometric complexity theory I: An approach to the P vs. NP and related problems. *SIAM Journal on Computing* 31, 2, (2001) 496–526.
30. Raghavendra, P. Optimal algorithms and inapproximability results for every csp? In *Proceedings of the 40th ACM Symposium on the Theory of Computing*. ACM, NY, 2008, 245–254.
31. Razborov, A. Lower bounds on the monotone complexity of some Boolean functions. *Soviet Mathematics-Doklady* 31, (1985) 485–493.
32. Razborov, A. On the method of approximations. In *Proceedings of the 21st ACM Symposium on the Theory of Computing*. ACM, NY, 1989, 167–176.
33. Razborov, A., and Rudich, S. Natural proofs. *Journal of Computer and System Sciences* 55, 1 (Aug. 1997), 24–35.
34. Shor, P. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing* 26, 5 (1997) 1484–1509.
35. Solovay, R. and Strassen, V. A fast Monte-Carlo test for primality. *SIAM Journal on Computing* 6 (1977), 84–85. See also erratum 7:118, 1978.
36. Sudan, M. Probabilistically checkable proofs. *Commun. ACM* 52, 3 (Mar. 2009) 76–84.
37. Trakhtenbrot, R. A survey of Russian approaches to Perebor (brute-force search) algorithms. *Annals of the History of Computing* 6, 4 (1984), 384–400.
38. Turing, A. On computable numbers, with an application to the Entscheidungs problem. *Proceedings of the London Mathematical Society* 42 (1936), 230–265.
39. van Melkebeek, D. A survey of lower bounds for satisfiability and related problems. *Foundations and Trends in Theoretical Computer Science* 2, (2007), 197–303.

Lance Fortnow (fortnow@eecs.northwestern.edu) is a professor of electrical engineering and computer science at Northwestern University's McCormick School of Engineering, Evanston, IL.

research highlights

P. 88

Technical Perspective Abstraction for Parallelism

By Katherine Yelick

P. 89

Optimistic Parallelism Requires Abstractions

By Milind Kulkarni, Keshav Pingali, Bruce Walter,
Ganesh Ramanarayanan, Kavita Bala, and L. Paul Chew

P. 98

Technical Perspective They Do Click, Don't They?

By Marc Dacier

P. 99

Spamalytics: An Empirical Analysis of Spam Marketing Conversion

By Chris Kanich, Christian Kreibich, Kirill Levchenko, Brandon Enright,
Geoffrey M. Voelker, Vern Paxson, and Stefan Savage

Technical Perspective

Abstraction for Parallelism

By Katherine Yelick

LOOKING FOR SOME new insight into an old problem? The following research paper by Milind Kulkarni et al. addresses the familiar problem of writing parallel applications and uses a fresh approach based on data abstraction to allow some challenging programs to be parallelized. Going back more than 30 years to the foundational work by Owicki and Gries on the semantics of parallel programs, and through decades of work on automatic parallelization of Fortran and C programs, the focus in the research and commercial communities has been on reasoning at the concrete level of read and write operations: if two iterations of a loop access the same variable, and at least one of them performs a write, then the iterations cannot execute in parallel. When combined with a static compile-time approach, the necessarily conservative analysis techniques mean that many loops cannot be parallelized, especially for programs with pointer-based data structures.

With a computer industry betting on multicore, the need for solutions to the parallel software problem has reached a new level of criticality. There has been a resurgence of parallelism research, much of it focused on dynamic discovery of parallelism using speculative techniques. The authors build on the idea of dynamic parallelism discovery by combining loop constructs with conflict analysis and a rollback mechanism to support speculative parallelism. The Galois system described in the paper has both ordered and unordered loops, but presents users with a serial semantics in both cases. Galois uses the type system to further control the behavior of such loops. Objects that use a traditional model of speculative parallelism are instances of so-called “catch-and-keep” classes, because the runtime system holds a lock through-

out an iteration to ensure that the iterations appear to execute serially.

But an interesting class of algorithms allow for correct behavior even in the presence of conflicting accesses. For example, branch and bound search can proceed in any order but must update the value of a variable representing the current bound. The authors add to speculative execution: programmers are allowed to specify that two method invocations on an object commute, meaning they can safely be reordered. In the branch and bound example, if a class is defined for the bound, with only operations to read the bound and update it monotonically by performing a max operation with a newly discovered bound, then the update operations commute with one another. Moreover, methods that have no effect on the abstract value, but “benevolent side effects” on the underlying state will commute at the abstract level despite having conflicting accesses.

Classes that have commutativity specifications are called “catch-and-release” classes, because the implementation holds a lock only during a method invocation to ensure atomicity of the operation, but not throughout the entire iteration. Serialization of the iterations is ensured instead through commutativity of the methods and, if abstract conflicts are discovered, though rollbacks.

The commutativity specification combined with unordered loops gives a programming model that is somewhere between explicit parallelism and sequential programming, as the programmer may give many opportunities for reordering operations, but the program still has a serial semantics. The one final concept needed to parallelize some complex irregular applications is the idea that methods with side effects

on catch-and-release classes must have inverse methods to “undo” the effects in case an abstract-level conflict is discovered. This allows iterations to execute in parallel, possibly with conflicting reads and writes to variables, as long as the methods involved are known to commute. If an abstract conflict is discovered between two method invocations that are not commutative, the runtime layer will roll back one of the iterations using the inverse of the operations that had been executed.

The authors successfully apply these ideas to two complex parallelization problems—an unstructured mesh refinement algorithm and a clustering algorithm used in data mining. Both problems involve irregular control flow and pointer-based data structures, making them intractable for static parallelism approaches or even speculative parallelism based on concrete conflict detection.

This paper presents the general ideas using these two compelling examples, and the concepts are both original and thought provoking. Abstraction mechanisms like object orientation are often considered deterrents to performance and parallelization, but in this approach data abstraction provides the specification point for commutativity and therefore a key to parallelization. The authors raise a number of interesting semantic issues in the examples and overall approach, and for those interested in a new perspective on parallel programming, I highly recommend this paper. **□**

Katherine Yelick is a professor of Electrical Engineering and Computer Sciences at the University of California, Berkeley, and the director of the National Energy Research Scientific Computing Center (NERSC), a national supercomputing facility at Lawrence Berkeley National Laboratory that serves the Department of Energy.

© 2009 ACM 0001-0782/09/0900 \$10.00

Optimistic Parallelism Requires Abstractions

By Milind Kulkarni, Keshav Pingali, Bruce Walter, Ganesh Ramanarayanan, Kavita Bala, and L. Paul Chew

Abstract

The problem of writing software for multicore processors is greatly simplified if we could automatically parallelize sequential programs. Although auto-parallelization has been studied for many decades, it has succeeded only in a few application areas such as dense matrix computations. In particular, auto-parallelization of irregular programs, which are organized around large, pointer-based data structures like graphs, has seemed intractable.

The Galois project is taking a fresh look at auto-parallelization. Rather than attempt to parallelize all programs no matter how obscurely they are written, we are designing programming abstractions that permit programmers to highlight opportunities for exploiting parallelism in sequential programs, and building a runtime system that uses these hints to execute the program in parallel. In this paper, we describe the design and implementation of a system based on these ideas. Experimental results for two real-world irregular applications, a Delaunay mesh refinement application and a graphics application that performs agglomerative clustering, demonstrate that this approach is promising.

1. INTRODUCTION

A pessimist sees the difficulty in every opportunity; an optimist sees the opportunity in every difficulty.

—Sir Winston Churchill

Irregular applications are organized around pointer-based data structures such as graphs and trees, and are ubiquitous in important application areas such as finite-elements, SAT solvers, maxflow computations, and compilers. In principle, it is possible to use a thread library (e.g., pthreads) or a combination of compiler directives and libraries (e.g., OpenMP) to write parallel code for irregular applications, but it is well known that writing explicitly parallel code can be very tricky because of the complexities of memory consistency models, synchronization, data races, etc. Tim Sweeney, who designed the multithreaded Unreal 3 game engine, estimates that writing multithreading code tripled software costs at Epic Games (quoted in de Galas³).

From the earliest days of parallel computing, it has been recognized that one way to circumvent the problems of writing explicitly parallel code is *auto-parallelization*.¹⁰ In this approach, application programmers write sequential programs, leaving it to the compiler or runtime system to extract and exploit the latent parallelism in programs. There is an enormous literature on algorithms and mechanisms for auto-parallelization, but like the characters in Pirandello's play *Six Characters in Search of an Author*, most

of them are in search of programs that they can parallelize. They can be divided into two categories: compile-time techniques and runtime techniques. Compile-time techniques use static analyses to find independent computations in programs, and have succeeded in parallelizing limited classes of irregular programs such as n-body methods.^{1, 5, 20} Runtime techniques use optimistic parallelization: computations are parallelized speculatively, and the runtime system detects conflicts between concurrent computations and rolls them back as needed to preserve the sequential semantics of the program. Optimistic parallelism is the basis of the popular Timewarp algorithm for parallel event-driven simulation,⁹ but efforts to build general-purpose systems based on optimistic parallelization, such as thread-level speculation (TLS),^{19, 22, 24} have had limited success. Because of these problems, interest in auto-parallelization has waned in recent years.

We are taking a fresh look at auto-parallelization, but from a different perspective than prior work in this area. Instead of trying to parallelize all application programs no matter how obscurely written, the Galois project is focusing on the following questions.

- Can we design sequential programming abstractions that capture the most commonly occurring parallelism patterns in programs?
- If so, what systems support is needed to auto-parallelize programs that use these abstractions?

A useful analogy is relational database programming. The SQL programmer views data as if they were organized as a flat table (relations), and operates on the data using high-level operations like joins and projections. Inside the database system, relations are implemented in very complex ways using B-trees, index structures, etc., and the high-level operations are performed in parallel using locks and transactions, but the relational abstractions enable these complications to be hidden from the SQL programmer.

Can we carry out a similar program for irregular applications? Although we are far from having a complete solution, the outlines of a solution for important patterns of parallelism are emerging from the fog. In this paper, we focus on understanding and exploiting parallelism in *iterative* irregular applications. In Section 2, we describe parallelism patterns in two such applications: a Delaunay mesh refinement

The original version of this paper appeared in the *Proceedings of the 2007 ACM SIGPLAN Conference on Programming Language Design and Implementation*.

code² and a graphics application²³ that performs agglomerative clustering.¹⁷ In Section 3, we discuss the Galois programming model and runtime system for exploiting this parallelism. In Section 4, we evaluate the performance of our system on the two applications. Finally, in Section 5, we discuss conclusions and ongoing work.

2. TWO IRREGULAR APPLICATIONS

In this section, we describe opportunities for exploiting parallelism in two irregular applications: Delaunay mesh refinement,² and agglomerative clustering¹⁷ as used within a graphics application.²³ These applications perform refinement and coarsening, respectively, which are arguably the two most common operations for bulk modification of irregular data structures.

2.1. Delaunay mesh refinement

The input to the 2D Delaunay mesh refinement algorithm is a triangulation of some region in the plane, in which all triangles satisfy a certain geometric property called the Delaunay condition.² Some of these triangles may be badly shaped according to certain geometric criteria; for example, excessively large triangles may cause unacceptable discretization errors in finite-element solutions. The goal of mesh refinement is to eliminate these badly shaped triangles from the mesh by replacing them with smaller triangles. However, performing this operation on a bad triangle may violate the Delaunay condition for neighboring triangles, so it is necessary to find all affected triangles (this is called the *cavity* of that bad triangle), and retriangulate the entire cavity. Figure 1 shows the initial mesh on the left (badly shaped triangles are colored black, and cavities are colored gray), and the refined mesh on the right. Refinement may create new badly shaped triangles, but there is a mathematical guarantee, at least in 2D, that if this process is repeated, a mesh without bad triangles will be produced in the end. The structure of the final mesh may depend on the order in which bad triangles are eliminated, but any mesh produced by this process is acceptable.

Figure 2 shows the pseudocode for mesh refinement. It is natural to organize the program around a work-list containing the bad triangles, as seen in lines 3 and 4. This work-list is one of the two key data structures in mesh refinement. The other is a graph representing the mesh structure; each triangle in the mesh is represented as one node, and edges in the graph represent triangle adjacencies in the mesh.

Figure 1. Fixing bad elements.

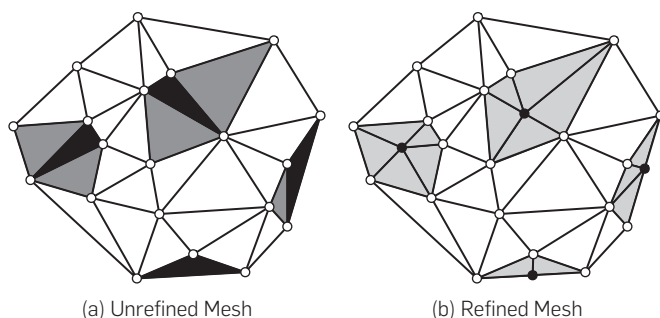


Figure 2. Pseudocode of the mesh refinement algorithm.

```

1: Mesh m = /* read in initial mesh */
2: WorkList wl ;
3: wl.add (mesh.badTriangles ());
4: while (wl.size () != 0) {
5:   Element e = wl.get (); //get bad triangle
6:   if (e no longer in mesh) continue;
7:   Cavity c = new Cavity (e);
8:   c.expand ();
9:   c.retriangulate ();
10:  mesh.update (c);
11:  wl.add (c.badTriangles ());
12: }

```

Opportunities for Exploiting Parallelism: Delaunay mesh refinement is a relatively complicated code since the central data structure is a graph that is modified repeatedly during the execution of the algorithm. Nevertheless, there may be a lot of parallelism in the algorithm since cavities that do not overlap can be processed in parallel, as in the mesh of Figure 1. If two cavities overlap, they must be processed sequentially in some order. How much parallelism is there in mesh refinement? Our studies¹¹ have shown that for a mesh of 100,000 triangles in which roughly half the initial triangles are badly shaped, there are more than 256 triangles that can be processed in parallel until almost the end of execution.

2.2. Agglomerative clustering

The second problem is *agglomerative clustering*, a well-known data-mining algorithm.¹⁷ This algorithm is used in graphics applications for handling large numbers of light sources.²³

The input to the clustering algorithm is (1) a data-set and (2) a measure of the similarity between items in the data-set. The goal of clustering is to construct a binary tree called a *dendrogram* whose hierarchical structure exposes the similarity between items in the data-set. Figure 3(a) shows a data-set containing points in the plane, for which the measure of similarity between data points is Euclidean distance. The dendrogram for this data-set is shown in Figure 3(b) and (c).

Agglomerative clustering can be performed by an iterative algorithm: at each step, the two closest points in the data-set are clustered together and replaced in the data-set by a single new point that represents the new cluster. The location of this new point may be determined heuristically.¹⁷ The algorithm terminates when there is only one point left in the data-set. Pseudocode for the algorithm is shown in Figure 4.

Figure 3. Agglomerative clustering.

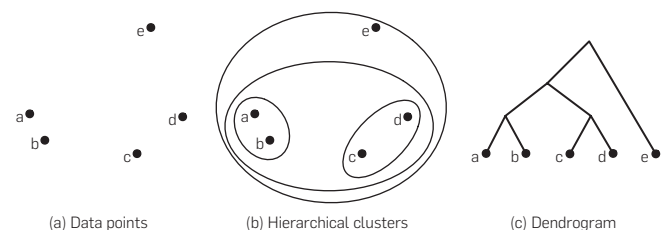


Figure 4. Pseudocode for agglomerative clustering.

```
1: kdTree := new KDTree (points)
2: pq := new PriorityQueue ( )
3: foreach p in points {pq.add(<p,kdTree.nearest(p)>)}
4: while (pq.size() != 0) do {
5:   Pair <p,n> := pq.get ( ); // return closest pair
6:   if (p.isAlreadyClustered ( ) ) continue;
7:   if (n.isAlreadyClustered ( ) ) {
8:     pq.add (<p,kdTree.nearest (p)>);
9:     continue;
10:  }
11:  Cluster c := new Cluster (p,n);
12:  dendrogram.add (c);
13:  kdTree.remove (p);
14:  kdTree.remove (n);
15:  kdTree.add (c);
16:  Point m:= kdTree . nearest (c);
17:  if (m != ptAtInfinity) pq.add (<c,m>);
18: }
```

The algorithm iterates over a priority queue whose entries are ordered pairs of points $\langle x,y \rangle$, such that y is the nearest neighbor of x (we call this $\text{nearest}(x)$). In each iteration of the while loop, the pair of points at the head of the priority queue—the closest pair—are clustered. These two points are replaced by a new, representative point. The nearest neighbor of this point is determined, and the pair is entered into the priority queue.

To find the nearest neighbor of a point, we can scan the entire data-set at each step, but this is too inefficient. Instead, we use a spatial acceleration structure called a *kd-tree* to find nearest neighbors. The kd-tree is built at the start of the algorithm and is kept up to date as points are removed and added from the space, as seen in Figure 4.

Opportunities for Exploiting Parallelism: Since each iteration clusters the two closest points in the current data-set, it may seem that the algorithm is inherently sequential. However, if we consider the data-set in Figure 3(a), we see that points a and b , and points c and d can be clustered concurrently since neither cluster affects the other. Intuitively, if the dendrogram is a long and skinny tree, there may be few independent iterations, whereas if the dendrogram is a bushy tree, there is parallelism that can be exploited since the tree can be constructed bottom-up in parallel. As in the case of Delaunay mesh refinement, the parallelism is very data-dependent. In experiments on graphics scenes with 20,000 lights, we have found that on average about 100 clusters can be constructed concurrently¹¹; thus, there is substantial parallelism that can be exploited.

2.3. Discussion

Existing compile-time parallelization techniques for irregular programs are based on *shape analysis*,²⁰ which determines structural invariants in the data structures. The graph in mesh refinement has no particular structure, and it is also modified in each iteration of the loop in Figure 2, so compile-time parallelization will not work for this application. Semi-static approaches using the *inspector-executor* model¹⁸ split computation into two phases: an inspector phase that

determines dependences between units of work and constructs a computation schedule and an executor phase that executes the resulting schedule in parallel. This approach does not work for mesh refinement since the dependence structure changes when the underlying graph is modified by the algorithm.

These considerations suggest that a fully dynamic approach in which dependences are detected at runtime is needed to parallelize codes like mesh refinement and agglomerative clustering. One such approach to parallelizing mesh refinement has been proposed by Hudson et al.,⁸ and it has the following steps: (1) compute the cavities of all bad triangles without making any modifications to the graph, (2) build an interference graph in which nodes represent cavities and edges represent overlapping cavities, (3) find a maximal independent set of nodes in this graph, and (4) retriangulate the cavities corresponding to these nodes in parallel, without any synchronization. These steps are then repeated for the new mesh until convergence. This approach can be viewed as an extended inspector-executor approach in which the execution of the inspector and executor are interleaved. However, this approach is very specific to Delaunay mesh refinement. For example, it is not clear that it can be used for applications like agglomerative clustering in which iterations are performed over priority queues.

3. THE GALOIS APPROACH

The analysis of Section 2 suggests that optimistic parallelization, in which computations are speculatively executed in parallel and rolled back selectively when dependence conflicts are detected by the runtime system, is the only general-purpose approach to exploiting parallelism in irregular applications. In this section, we argue that optimistic parallelization needs to be coupled with appropriate programming abstractions, and we describe an implementation of these ideas in the Galois system.

The need for programming abstractions becomes evident if we consider the mesh refinement code in Figure 2. The work-list of bad triangles determines a particular order in which bad triangles are processed by the sequential program, and any auto-parallelized version of this code will be forced to process bad triangles in the same order. The fact that bad triangles can actually be processed in any order is important for parallelization, but it is missing from this code. Abstractly, we can view the processing of each bad triangle as an operator that is applied to the graph to modify a small region of it; the fact that bad triangles can be processed in any order is equivalent to asserting that the applications of this operator to the graph “commute” with each other. Since the structure of the final graph may actually be different for different operator orderings, we call this *application-specific commutativity* for obvious reasons.

Opportunities to exploit commutativity may also arise in abstract data type (ADT) implementations. Consider a set ADT that is implemented using a linked list. With respect to the semantics of sets, insert operations commute with each other since all insertion orders produce the same set even though the linked list representation internal to the ADT may be different for different insertion orders. In this case,

commutativity arises from the fact that there may be several concrete (memory) states that represent a single abstract state. Exploiting this kind of *ADT commutativity* obviously requires an object-oriented language.

We will refer to application-specific and ADT commutativity as *semantic commutativity*. In contrast, traditional compile-time parallelization techniques such as dependence analysis¹⁰ and Diniz and Rinard's commutativity analysis⁴ focus on *concrete commutativity* in which all orders of performing operations result in the same concrete state. Semantic commutativity is more general, and it permits more interleavings of operations.

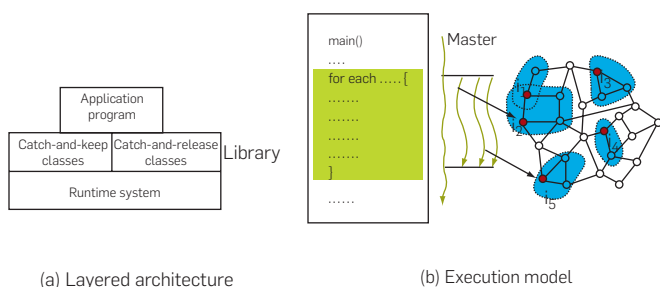
The programming abstractions introduced in this section permit programmers to highlight opportunities for exploiting semantic commutativity. They can be added to any sequential object-oriented language (the results in this paper are from an implementation in C++). Figure 5(a) is a conceptual picture of the Galois system. Application programs use two constructs called *Galois set iterators*, described in Section 3.1, for highlighting opportunities for exploiting parallelism. Section 3.2 describes the data structure library. Data structures in which there are opportunities to exploit ADT commutativity are implemented by *catch-and-release* classes, while others are implemented in *catch-and-keep* classes. The runtime system implements optimistic parallelization, and detects and recovers from potentially unsafe accesses to shared objects, as explained in Section 3.3.

3.1. Galois set iterators

The Galois programming model is sequential and object-oriented; programs are written in an object-oriented language like C++ or Java extended with two constructs called Galois set iterators.

- Unordered-set iterator: for each e in set S do B(e)**
 The loop body B(e) is executed for each element e of set S. Since set elements are not ordered, this construct asserts that in a serial execution of the loop, the iterations can be executed in any order. There may be dependences between the iterations, as in the case of Delaunay mesh refinement, but any serial order of executing iterations is permitted. When an iteration executes, it may add elements to S.
- Ordered-set iterator: for each e in Poset S do B(e)**
 This construct iterates over a partially ordered set (Poset) S. It is similar to the set iterator above, except that any execution order must respect the partial order imposed by the Poset S.

Figure 5. The Galois system.



Note that new elements may be added to a set while iterating over it, which is not allowed in conventional set iterators in languages like SETL or Java. Figure 6 shows the client code for Delaunay mesh generation. Instead of a work-list of bad triangles, this code uses a set of bad triangles and a set iterator. Since set elements are not ordered, the iterator is permitted to iterate over the set in any order. Therefore, the Galois version makes evident the fact that the bad triangles can be processed in *any* order, a fact that is absent from the more conventional code of Figure 2. For lack of space, we do not show the Galois version of agglomerative clustering, but it uses the ordered-set iterator in the obvious way.

The parallel execution model is shown in Figure 5(b). A master thread executes all code outside the Galois set iterators. When it encounters a Galois set iterator, it enlists worker threads to help execute iterations concurrently. The assignment of iterations to threads is done dynamically, but this policy can be changed by an expert programmer.¹² Threads synchronize by barrier synchronization at the end of the iterator.

Given this execution model, the main technical problem is to ensure that the parallel execution respects the sequential semantics of the iterators. For an unordered-set iterator, this can be accomplished by ensuring that the execution of each iteration has *transactional semantics*. These semantics guarantee *serializability*; the parallel execution will behave as if the iterations were executed serially in some order. For the ordered-set iterator, we must also ensure that the iterations appear to execute in the order prescribed by the ordering on set elements. Guaranteeing sequential semantics is a nontrivial problem because each iteration may read and write objects in shared memory, so we must ensure that these reads and writes are properly coordinated. Next, we describe how this is accomplished.

3.2. Galois library classes

The library has two kinds of classes: catch-and-keep and catch-and-release. As in Java, a lock is automatically associated with each object, but the locking policy is different for the two kinds of classes.

Catch-and-Keep Classes: Catch-and-keep classes are the default, and they are implemented in Galois using a variation of the well-known two-phase locking policy. To invoke a method on a catch-and-keep object, an iteration must

Figure 6. Delaunay mesh refinement using set iterator.

```

1: Mesh m = /* read in initial mesh */
2: Set wl;
3: wl.add (mesh.badTriangles ());
4: for each e in wl do {
5:     if (e no longer in mesh) continue;
6:     Cavity c = new Cavity (e);
7:     c.expand ();
8:     c.retriangulate ();
9:     m.update (c);
10:    wl.add (c.badTriangles ());
11: }
    
```

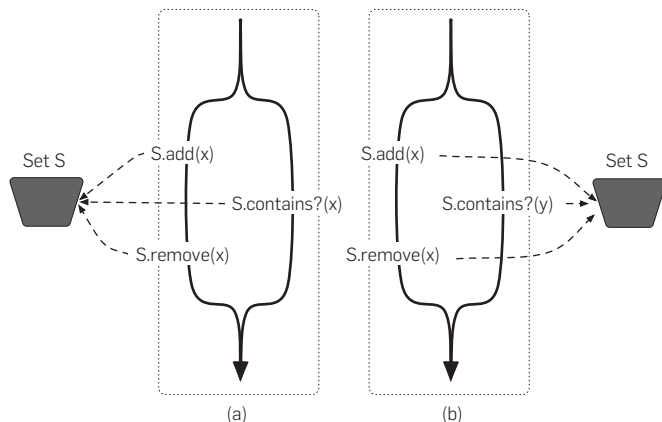
first acquire the lock associated with it. This lock is held until the iteration terminates, at which point the iteration releases all of its locks. If an iteration is unable to acquire a lock on an object, this means that a second iteration is currently accessing the object, and one of the two iterations must be rolled back. Rollbacks are accomplished by copying an object before it is modified, and restoring from that copy upon rollback. Thus, in a system in which all objects use the catch-and-keep policy, serialization of iterations is easy to ensure. Acquiring and releasing locks, making backup copies of objects, etc., is performed automatically by our runtime system, as explained in Section 3.3. It is also possible to use hardware transactional memory (TM).⁷

While catch-and-keep classes are simple to implement, they may not provide enough concurrency. Work-sets are themselves data structures, and they are implemented using classes in the Galois library. Since each iteration gets an element from the work-set at the beginning and may add elements to it at the end, a catch-and-keep implementation of the work-set class would permit only one iteration to execute at a time.

Catch-and-Release Classes: To solve this problem, the Galois system supports the catch-and-release policy for concurrently accessed objects such as the graph and work-set in mesh refinement, or the kd-tree in agglomerative clustering. To access a catch-and-release object, an iteration must acquire the lock on the object. However, unlike in catch-and-keep, the lock is released as soon as the method completes. Releasing the lock allows interleaving of method invocations from different iterations, which increases concurrency.

The key problem in catch-and-release is to ensure that the method interleavings do not violate serializability of iterations. This is nontrivial, as demonstrated by the programs in Figure 7, which show iterations manipulating a set that supports `add`, `remove`, and `contains`, with the standard semantics. In Figure 7(a), we see that in any sequential execution, the call `contains(x)` will return false. However, for the interleaving shown in the figure, the call will return true. On the other hand, for the program in Figure 7(b), all possible interleavings of the methods match a serial execution. For a catch-and-release object,

Figure 7. Interleaving method invocations from different iterations.



how can we determine which interleavings are legal, and which should be disallowed?

The key is semantic commutativity, described at the beginning of this section. Method invocations to a given object from two iterations can be interleaved safely provided that the final abstract state is consistent with some serial order of iteration execution. In Figure 7(a), the invocation `contains(x)` does not commute with the operations from the other thread, so the invocations from the two iterations must not be interleaved. In Figure 7(b), `contains(y)` commutes with the other operations, so the iterations can execute in parallel. Note that commutativity may depend on the arguments or return values of methods.

Because iterations are executed in parallel, it is possible for commutativity conflicts to prevent an iteration from completing, requiring that iterations be rolled back. Because semantic commutativity does not track the concrete state of an object, simply creating copies of the concrete state (as in catch-and-keep classes) does not suffice. Instead, every method of a catch-and-release object that may modify the state of that object must have an associated *inverse* method that undoes the side-effects of that method invocation. For example, for a set that does not contain x , the inverse of a method invocation that adds an element x to a set is a method invocation that removes it from that set. As in the case of commutativity, what is relevant for our purpose is an inverse in the *semantic* sense; invoking a method and its inverse in succession may not restore the concrete data structure to what it was.

Note that when an iteration rolls back, all of the methods which it invokes during roll-back must succeed. Thus, we must never encounter conflicts when invoking inverse methods. When the Galois system checks commutativity, it also checks commutativity with the associated inverse method. *Putting It All Together:* ADT commutativity and undo must be specified by the class designer. Figure 8 illustrates how

Figure 8. Example Galois class for a set.

```
class Set {
    //interface methods
    void add (Element x);
    [commutes]
    - add (y) {y != x}
    - remove (y) {y != x}
    - contains (y) {y != x}
    [inverse] remove (x)
    void remove (Element x);
    [commutes]
    - add (y) {y != x}
    - remove (y) {y != x}
    - contains (y) {y != x}
    [inverse] add (x)
    boolean contains (Element x);
    [commutes]
    - add (y) {y != x}
    - remove (y) {y != x}
    - contains (*)//any call to contains
}
}
```

this information is specified in Galois for a class that implements sets. For each method, the implementor specifies the following:

- *Commutes*: This section specifies which other methods the current method commutes with, and under which conditions. For example, `remove(x)` commutes with `add(y)`, as long as the elements are different.
- *Inverse*: This section specifies the inverse of the current method.

Note that `add(x)` does not commute with `add(x)` according to this specification. This is because rolling back `add(x)` requires invoking `remove(x)`, which would conflict with other invocations of `add(x)`. This choice simplifies the implementation.

3.3. Runtime system

The Galois runtime system has two components: (1) a global structure called the *commit pool* that is responsible for creating, aborting, and committing iterations and (2) structures called *conflict logs* which detect when commutativity conditions are violated for catch-and-release objects.

The commit pool maintains an *iteration record*, shown in Figure 9, for each ongoing iteration in the system. The status of an iteration can be `RUNNING`, `RTC` (ready-to-commit), or `ABORTED`. Threads go to the commit pool to obtain an iteration. The commit pool creates a new iteration record, obtains the next element from the iterator, assigns a priority to the iteration record based on the priority of the element (for a set iterator, all elements have the same priority), and sets the status field of the iteration record to `RUNNING`. When an iteration invokes a method of a shared object, (1) the conflict log of that object is updated, as described in more detail below and (2) a callback to the associated inverse method is pushed onto the undo log of the iteration record. If a commutativity conflict is detected, the commit pool arbitrates between the conflicting iterations, and aborts iterations to permit the highest priority iteration to continue execution. Callbacks in the undo logs of aborted iterations are executed to undo their effects on shared objects. Once a thread has completed an iteration, the status field of that iteration is changed to `RTC`, and the thread is allowed to begin a new iteration. When the completed iteration has the highest priority in the system, it is allowed to commit. It can be seen that the role of the commit pool is similar to that of a reorder buffer in out-of-order processors.

Figure 9. Iteration record maintained by runtime system.

```
IterationRecord {
    Status status;
    Priority p;
    UndoLog ul;
    Lock l;
}
```

Conflict Logs: The *conflict log* is the mechanism for detecting commutativity conflicts. There is one conflict log associated with each catch-and-release object. A simple implementation for the conflict log of an object is a list containing the method signatures (including the values of the input and output parameters) of all invocations on that object made by currently executing iterations (called “outstanding invocations”). When iteration i attempts to call a method m_1 on an object, the method signature is compared against all the outstanding invocations in the conflict log. If one of the entries in the log does not commute with m_1 , then a commutativity conflict is detected, and an arbitration process is begun to determine which iterations should be aborted, as described below. If m_1 commutes with all the entries in the log, the signature of m_1 is appended to the log. When i either aborts or commits, all the entries in the conflict log inserted by i are removed from the conflict log.

Consider the effects of calling `contains(x)` on a set implementing the interface shown in Figure 8. The conflict log contains all the outstanding invocations of methods on the set. Because `contains` can only conflict with `add` or `remove`, the runtime system will scan the log to ensure that no other iteration called `add(x)` or `remove(x)`.

Note that for efficiency, a runtime system may use an optimized implementation of conflict logs which does not require a full scan of all outstanding method invocations to detect conflicts. The full version of this paper describes a number of these optimizations in more detail.¹⁴

Commit Pool: When an iteration attempts to commit, the commit pool checks two things: (1) that the iteration is at the head of the commit queue, and (2) that the priority of the iteration is higher than all the elements left in the set/Poset being iterated over. If both conditions are met, the iteration can successfully commit. If the conditions are not met, the iteration must wait until it has the highest priority in the system; its status is set to `RTC`, and the thread is allowed to begin another iteration.

When an iteration successfully commits, the thread that was running that iteration also checks the commit queue to see if more iterations in the `RTC` state can be committed. If so, it commits those iterations before beginning the execution of a new iteration. When an iteration has to be aborted, the status of its record is changed to `ABORTED`, but the commit pool takes no further action. Such iteration objects are lazily removed from the commit queue when they reach the head.

Conflict Arbitration: The other responsibility of the commit pool is to arbitrate conflicts between iterations. When iterating over an unordered set, the choice of which iteration to roll back in the event of a conflict is irrelevant. For simplicity, we always choose the iteration which detected the conflict. However, when iterating over an ordered set, the lower priority iteration must be rolled back while the higher priority iteration must continue. Without doing so, there exists the possibility of deadlock. Thus, when iteration i_1 calls a method on a shared object and a conflict is detected with iteration i_2 , the commit pool arbitrates based on the priorities of the two iterations. If i_1 has lower

priority, it simply performs the standard rollback operations. The thread which was executing i_1 then begins a new iteration.

This situation is complicated when i_2 is the iteration that must be rolled back. Because the Galois runtime functions at the user-level, there is no way to roll back an iteration running on another thread. Instead, i_1 undoes the effects of i_2 without explicitly rolling back execution. Next, i_1 sets a flag on i_2 telling it to roll back. When the thread running i_2 invokes a shared method or attempts to commit, it checks this flag and completes the rollback.

When an iteration has to be aborted, the callbacks in its undo log are executed in LIFO order. Note that the arguments used by the callback must have the values present when the callback was created. This is ensured due to the LIFO ordering of the undo log, as any later changes to the arguments will be undone first.

3.4. Discussion

There is no analog of unordered-set iterators or catch-and-release objects in current TLS systems^{22, 24} (in fact, most of these systems auto-parallelize programs in FORTRAN and C, which have no notion of data abstraction). It is possible that this might account for the limited performance of these systems.

The TM paper of Herlihy and Moss⁷ has inspired a vast literature on transactions and TM (see Larus and Rajwar¹⁵ for a survey of the more important results). The starting point for the transactional approach is an explicitly parallel program, and the focus is on reducing the complexities and overhead of synchronization through the use of the transactional model. In contrast, our starting point is a sequential program, and the focus is on auto-parallelization. The Galois runtime system exploits optimistic parallelism just as TM exploits optimistic synchronization. Hardware TM can be used to implement catch-and-keep classes with low overhead, but catch-and-release classes must be supported in software. Herlihy and Koskinen have recently introduced catch-and-release objects into a software TM to *boost* its performance.⁶

Ni et al.¹⁶ have proposed to extend the conventional transactional model with open nested transactions and abstract locking to allow more abstract conflict checking. Open nesting is a *mechanism*, and it does not specify how the abstract locks should be used. Semantic commutativity provides the appropriate definition of semantic conflict for data structures, and open nesting is one possible means of implementing semantic commutativity.

4. EVALUATION

Our initial implementation of the Galois system was in C++. Our evaluation platform was a 4 processor, 1.5 GHz Itanium 2, with 16KB of L1, 256KB of L2 and 3MB of L3 cache per processor. The threading library was pthreads.

4.1. Delaunay mesh refinement

We first wrote a sequential Delaunay mesh refinement program without locks, threads etc. to serve as a *reference* implementation. We then implemented a Galois version (which we

call *meshgen*), as well as an explicitly parallel, fine-grain locking program (*FGL*) that uses locks on individual triangles. The Galois version uses the set iterator, and the runtime system described in Section 3.3. In all three implementations, the mesh was represented by a graph that was implemented as a set of triangles, where each triangle maintained a set of its neighbors. For *meshgen*, code for commutativity checks was added by hand to this graph class; ultimately, we would like to generate this code automatically from high-level commutativity specifications like those in Figure 8. We used an STL queue to implement the work-set. We refer to these default implementations of *meshgen* and *FGL* as *meshgen(d)* and *FGL(d)*.

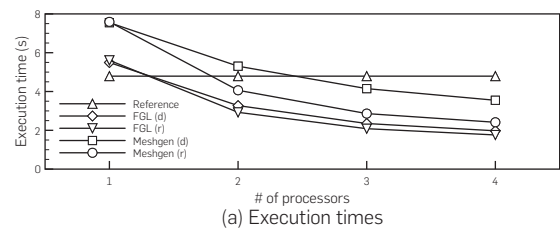
To understand the effect of scheduling policy on performance, we implemented two more versions, *FGL(r)* and *meshgen(r)*, in which the work-set was implemented by a data structure that returned a random element of the work-set.

The input data-set was generated automatically using Jonathan Shewchuk's Triangle program.²¹ It had 10,156 triangles and boundary segments, of which 4,837 triangles were bad.

Execution Times and Speedups: Execution times for the five implementations on the Itanium machine are shown in Figure 10(a). The reference version is the fastest on a single processor. On four processors, *FGL(d)* and *FGL(r)* differ only slightly in performance. *Meshgen(r)* performed almost as well as *FGL*, although surprisingly, *meshgen(d)* was twice as slow as *FGL*.

Statistics on Committed and Aborted Iterations: To understand these issues better, we determined the total number of committed and aborted iterations for different versions of *meshgen*, as shown in Figure 10(b). On one processor, *meshgen* executed and committed 21,918 iterations. Because of the inherent nondeterminism of the set

Figure 10. Mesh refinement results.



# of proc.	Committed			Aborted		
	Max	Min	Avg	Max	Min	Avg
1	21918	21918	21918	n/a	n/a	n/a
4 (meshgen(d))	22128	21458	21736	28929	27711	28290
4 (meshgen(r))	22101	21738	21909	265	151	188

(b) Committed and aborted iterations for *meshgen*

Source of overhead	% of overhead
Abort	10
Commit	10
Scheduler	3
Commutativity	77

(c) Breakdown of Galois overhead for *meshgen(r)*

iterator, the number of iterations executed by meshgen in parallel varies from run to run (the same effect will be seen on one processor if the scheduling policy is varied). Therefore, we ran the codes a large number of times, and determined a distribution for the numbers of committed and aborted iterations. Figure 10(b) shows that on four processors, meshgen(d) committed roughly the same number of iterations as it did on one processor, but also aborted almost as many iterations due to cavity conflicts. The abort ratio for meshgen(r) is much lower because the scheduling policy reduces the likelihood of conflicts between processors. The lower abort ratio accounts for the performance difference between meshgen(d) and meshgen(r). Because the FGL code is carefully tuned by hand, the cost of an aborted iteration is substantially less than the corresponding cost in meshgen, so FGL(r) performs only a little better than FGL(d).

It seems counterintuitive that a randomized scheduling policy could be beneficial, but a deeper investigation into the source of cavity conflicts showed that the problem could be attributed to our use of an STL queue to implement the work-set. When a bad triangle is refined by the algorithm, a cluster of smaller bad triangles may be created within the cavity. In the queue data structure, these new bad triangles are adjacent to each other, so it is likely that they will be scheduled together for refinement on different processors, leading to cavity conflicts. One conclusion from these experiments is that domain knowledge is invaluable for implementing a good scheduling policy.

Overhead Breakdown: The Galois system introduces some overhead over the reference code, even when running on one processor; meshgen(r) takes 58% longer to execute the reference code on the same input. To understand the overheads of the Galois implementations, we instrumented the code using PAPI. We broke down the Galois overhead into four categories: (1) commit overhead, (2) abort overhead, (3) scheduler overhead, which includes time spent arbitrating conflicts, and (4) commutativity check overhead. The results, as seen in Figure 10(c), show that roughly three fourths of the Galois overhead goes in performing commutativity checks. It is clear that reducing this overhead is key to reducing the overall overhead of the Galois runtime.

4.2. Agglomerative clustering

For the agglomerative clustering problem, the two main data structures are the kd-tree and the priority queue. The kd-tree interface is essentially the same as set, but with the addition of the nearest neighbor (*nearest*) method. The priority queue is an instance of a Poset. Since the priority queue is used to sequence iterations, the removal and insertion operations (*get* and *add*, respectively) are orchestrated by the commit pool.

To evaluate the agglomerative clustering algorithm, we modified an existing graphics application called *lightcuts* that provides a scalable approach to illumination.²³ The code builds a light hierarchy based on a distance metric that factors in Euclidean distance, light intensity, and light direction. We modified the objects used in the light

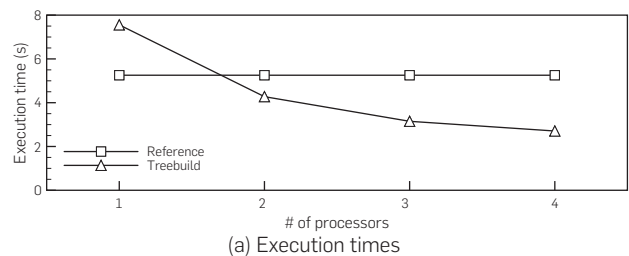
clustering code to use Galois interfaces and the Poset iterator for tree construction. The overall structure of the resulting code was discussed in Figure 4. We will refer to this Galois version as *treebuild*. We compared the running time of *treebuild* against a sequential *reference* version.

Figure 11 gives the performance results. These results are similar to the Delaunay mesh generation results discussed in Section 4.1, so we describe only the points of note. The execution times in Figure 11(a) show that despite the serial dependence order imposed by the priority queue, the Galois system is able to expose a significant amount of parallelism. The mechanism that allows us to do this is the commit pool, which allows threads to begin execution of iterations even if earlier iterations have yet to commit. The overhead introduced by the Galois system is 44% on a single processor. We see that due to the overhead of managing the commit pool, the scheduler accounts for a significant percentage of the overall Galois overhead, as seen in Figure 11(c).

4.3. Ongoing work

In recent work, we introduced the notion of logical *data partitioning* into the Galois system.¹³ For mesh refinement, each partition of the graph is mapped to a core, and each core processes bad triangles in its own partition. This mapping reduces the likelihood of conflicts since different cores work in different regions of the graph; unlike the randomized schedule discussed in Section 4, this approach also promotes locality of reference. Furthermore, commutativity checks, which are expensive, can be replaced with locking on partitions. Over-decomposition of the graph increases the likelihood that a core will have work to do even if some of its partitions are locked by other cores

Figure 11. Agglomerative clustering results.



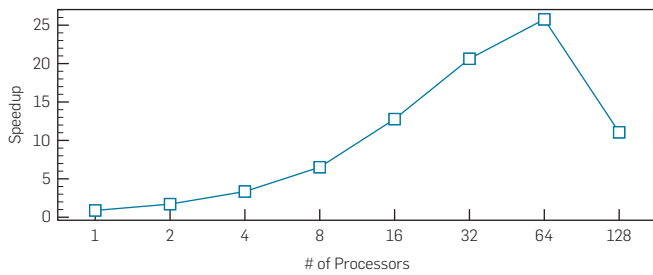
# of proc.	Committed			Aborted		
	Max	Min	Avg	Max	Min	Avg
1	57846	57846	57846	n/a	n/a	n/a
4	57870	57849	57861	3128	1887	2528

(b) Committed and aborted iterations in treebuild

Source of overhead	% of overhead
Abort	1
Commit	8
Scheduler	39
Commutativity	52

(c) Breakdown of Galois overhead

Figure 12. Speedup vs. # of processors for mesh refinement.




working on cavities that span multiple partitions. However, load-balancing is more of a problem than in the baseline approach. We also developed a scheduling framework that gives programmers control over the scheduling policy used by the Galois runtime.¹²

We produced a new implementation of the Galois system in Java, which incorporates these changes. We then evaluated an implementation of mesh refinement, using an input mesh of 100,000 triangles. Figure 12 shows the performance of this implementation on a 128-core Sunfire system, normalized to a sequential implementation in plain Java. We see that the Galois system is able to achieve significant speedup up to 64 cores, beyond which load imbalance and communication latency begin to dominate performance.

5. CONCLUSION

In this paper, we described the Galois system, which is a fresh approach to automatic parallelization of irregular applications. Rather than attempt to parallelize all programs no matter how obscurely they are written, our system provides programming abstractions that programmers use to highlight opportunities for exploiting parallelism. The runtime system uses optimistic parallelization to exploit these opportunities for parallel execution highlighted by the programmer. It detects conflicts between concurrent computations, and rolls back computations appropriately to preserve the sequential semantics of the program. Experimental results for two real-world irregular applications, a Delaunay mesh refinement application and a graphics application that performs agglomerative clustering, demonstrate that this approach is promising.

The Galois approach should be viewed as a baseline parallel implementation for irregular applications. Handwritten parallel versions of many irregular applications exploit particular kinds of structure in these applications to reduce parallel overheads. How do we identify such opportunities for exploiting structure in irregular programs? Can the relevant optimizations be performed automatically by the compiler? How do we reduce runtime overheads? These are some of the exciting research opportunities that lie ahead.

This work is supported in part by NSF grants 0833162, 0719966, 0702353, 0724966, 0739601, and 0615240, as well as grants from IBM and Intel Corporation. 

References

- Burke, M., Carini, P., Choi, J.-D. *Interprocedural Pointer Alias Analysis*. Technical Report IBM RC 21055, IBM Yorktown Heights, 1997.
- Chew, L.P. Guaranteed-quality mesh generation for curved surfaces. In *SCG'93: Proceedings of the 9th Annual Symposium on Computational Geometry* (1993), 274–280.
- de Galas, J. The quest for more processing power: is the single core CPU doomed? <http://www.anandtech.com/cpuchipsets/showdoc.aspx?I=2377>, February 2005.
- Diniz, P.C., Rinard, M.C. Commutativity analysis: a new analysis technique for parallelizing compilers. *ACM Trans. Prog. Lang. Syst.* 19, 6 (1997), 942–991.
- Ghiya, R., Hendren, L. Is it a tree, a dag, or a cyclic graph? A shape analysis for heap-directed pointers in c. In *POPL*, 1996.
- Herlihy, M., Koskinen, E. Transactional boosting: a methodology for highly-concurrent transactional objects. In *Principles and Practices of Parallel Programming (PPoPP)*, 2008.
- Herlihy, M., Moss, J.E.B. Transactional memory: architectural support for lock-free data structures. In *ISCA '93: Proceedings of the 20th Annual International Symposium on Computer Architecture* (1993).
- Hudson, B., Miller, G.L., Phillips, T. Sparse parallel Delaunay mesh refinement. In *SPAA* (2007).
- Jefferson, D.R. Virtual time. *ACM Trans. Prog. Lang. Syst.* 7, 3 (1985), 404–425.
- Kennedy, K., Allen, J., editors. *Optimizing Compilers for Modern Architectures: A Dependence-Based Approach*. Morgan Kaufmann, 2001.
- Kulkarni, M., Burtscher, M., Inkulu, R., Pingali, K., Cascaval, C. How much parallelism is there in irregular applications? In *Principles and Practices of Parallel Programming (PPoPP)*, 2009.
- Kulkarni, M., Carribault, P., Pingali, K., Ramanarayanan, G., Walter, B., Bala, K., Chew, L.P. Scheduling strategies for optimistic parallel execution of irregular programs. In *Symposium on Parallel Architectures and Algorithms (SPAA)* (2008).
- Kulkarni, M., Pingali, K., Ramanarayanan, G., Walter, B., Bala, K., Chew, L.P. Optimistic parallelism benefits from data partitioning. *SIGARCH Comput. Archit. News* 36, 1 (2008), 233–243.
- Kulkarni, M., Pingali, K., Walter, B., Ramanarayanan, G., Bala, K., Chew, L.P. Optimistic parallelism requires abstractions. *SIGPLAN Not (Proceedings of PLDI 2007)* 42, 6 (2007), 211–222.
- Larus, J., Rajwar, R. *Transactional Memory (Synthesis Lectures on Computer Architecture)*. Morgan & Claypool Publishers, 2007.
- Ni, Y., Menon, V., Adl-Tabatabai, A.-R., Hosking, A.L., Hudson, R., Moss, J.E.B., Saha, B., Shpeisman, T. Open nesting in software transactional memory. In *Principles and Practices of Parallel Programming (PPoPP)*, 2007.
- Pang-Ning Tan, M.S., Kumar, V., editors. *Introduction to Data Mining*. Pearson Addison Wesley, 2005.
- Ponnusamy, R., Saltz, J., Choudhary, A. Runtime compilation techniques for data partitioning and communication schedule reuse. In *Proceedings of the 1993 ACM/IEEE Conference on Supercomputing* (1993).
- Rauchwerger, L., Padua, D.A. The LRPD test: Speculative run-time parallelization of loops with privatization and reduction parallelization. *IEEE Trans. Parallel Distrib. Syst.* 10, 2 (1999), 160–180.
- Sagiv, M., Reps, T., Wilhelm, R. Solving shape-analysis problems in languages with destructive updating. In *Proceedings of the 23rd Annual ACM Symposium on the Principles of Programming Languages* (St. Petersburg Beach, FL, January 1996).
- Shewchuk, J.R. Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator. In *Applied Computational Geometry: Towards Geometric Engineering*, volume 1148 of *Lecture Notes in Computer Science*. May 1996, 203–222.
- Steffan, J.G., Colohan, C.B., Zhai, A., Mowry, T.C. A scalable approach to thread-level speculation. In *ISCA '00: Proceedings of the 27th Annual International Symposium on Computer Architecture* (2000).
- Walter, B., Fernandez, S., Arbree, A., Bala, K., Donikian, M., Greenberg, D. Lightcuts: a scalable approach to illumination. *ACM Trans. Graphics (SIGGRAPH)* 24, 3 (July 2005), 1098–1107.
- Zhan, L.R.Y., Torrellas, J. Hardware for speculative run-time parallelization in distributed shared-memory multiprocessors. In *HPCA '98: Proceedings of the 4th International Symposium on High-Performance Computer Architecture* (1998).

Milind Kulkarni and Keshav Pingali
({milind,pingali}@cs.utexas.edu),
University of Texas, Austin.

Bruce Walter, Ganesh Ramanarayanan,
Kavita Bala, and L. Paul Chew
(bjw@graphics.cornell.edu,
{graman,kb,chew}@cs.cornell.edu),
Cornell University, Ithaca, NY.

Technical Perspective

They Do Click, Don't They?

By Marc Dacier

YOU NEVER CLICK on advertisements received in spam or in phishing messages, do you? Of course you don't! Nobody does. At least, that's the typical response one hears. So, if that is true, why are we still getting an enormous amount of unsolicited email messages? How can this advertisement business be profitable if no one follows any links; if nobody buys anything?

The following paper by Chris Kanich et al. addresses these issues in very concrete terms, thanks to an impressive experiment that enables the authors to offer us to look at some real-world spam campaigns from the inside. This is a fascinating piece of

This is a fascinating piece of work. Not only does it help debunk some unscientific claims related to the underground economy, but also, and more importantly, it is very likely to become a seminal reference for a new area of research.

work. Not only does it help debunk some unscientific claims related to the underground economy, but also, and more importantly, it is very likely to become a seminal reference for a new area of research.

This work could have led to a disaster. The authors could have opened Pandora's Box by infiltrating a botnet as they did. The dark side of the force is so strong. Fortunately, these authors are well known for their ability to carry out scientific experiments with all the rigor, precision, and discipline required. They have taken great care addressing the legal and ethical issues linked to the measurement campaign they wanted to carry out. As a result, this paper is a must-read for all those who will be tempted in the future to assess quantitatively the various Internet threats or the motivations and the modus operandi of the organizations launching daily attacks. I do sincerely hope this work will stimulate other teams to carry out more, and similar, experimental work.

"Security by obscurity," that is, keeping vulnerabilities secret in the hope that malicious actors will never find them, is fortunately a concept of the past. However, 15 years ago the issue was still controversial. Today, numerous forums exist where information is shared on the latest exploits, tools, and techniques to break in to systems. But, it is still rare to see someone openly discussing, in very precise terms, the dynamics of these threats. Who is doing what, how often, against whom, why? Few actors have unbiased and usable information on these topics. Those who have such a goldmine are usually unable (for legal reasons) or unwilling (to preserve some competitive advantage) to describe their

assets and the lessons they learn when mining them. Experiments such as the one presented in this paper highlight the fact that it is possible, even within the context of a limited experiment, to learn a lot about these hidden markets. Of course, I am not underestimating the amount of effort the authors have invested in this study. Indeed, if anyone else has a great idea for measuring the negative forces we are facing on the Net, if they define their experiments very carefully, and are really cautious when interpreting their results, then they can also contribute to a better understanding of the malware economy.

As computer scientists in general, and computer security researchers in particular, we do not have a long tradition of running and presenting experiments in such a way that others can repeat. This should change. Other scientific communities do much better than us and we should probably learn from them. The following paper offers a unique opportunity to start working in that direction. Its contributions are twofold. First, it gives us new insight on the spam market, its dynamics, its actors, and so on. Second, the precise presentation of the whole experimental process—before, during, and after the experiment itself—is a masterpiece of how to do things the right way. This second contribution is probably as important, if not more, than the first in a domain where the greatest care must be taken. Before being tempted to play the sorcerer's apprentice, you should definitely read this paper. **□**

Marc Dacier is the director of the Symantec Research Labs in Europe.

© 2009 ACM 0001-0782/09/0900 \$10.00

Spamalytics: An Empirical Analysis of Spam Marketing Conversion

By Chris Kanich, Christian Kreibich, Kirill Levchenko, Brandon Enright, Geoffrey M. Voelker, Vern Paxson, and Stefan Savage

Abstract

The “conversion rate” of spam—the probability that an unsolicited email will ultimately elicit a “sale”—underlies the entire spam value proposition. However, our understanding of this critical behavior is quite limited, and the literature lacks any quantitative study concerning its true value. In this paper we present a methodology for measuring the conversion rate of spam. Using a parasitic infiltration of an existing botnet’s infrastructure, we analyze two spam campaigns: one designed to propagate a malware Trojan, the other marketing online pharmaceuticals. For nearly a half billion spam emails we identify the number that are successfully delivered, the number that pass through popular antispam filters, the number that elicit user visits to the advertised sites, and the number of “sales” and “infections” produced.

1. INTRODUCTION

Spam-based marketing is a curious beast. We all receive the advertisements—“Excellent hardness is easy!”—but few of us have encountered a person who admits to following through on this offer and making a purchase. And yet, the relentlessness by which such spam continually clogs Internet inboxes, despite years of energetic deployment of antispam technology, provides undeniable testament that spammers find their campaigns profitable. Someone is clearly buying. But how many, how often, and how much?

Unraveling such questions is *essential* for understanding the economic support for spam and hence where any structural weaknesses may lie. Unfortunately, spammers do not file quarterly financial reports, and the underground nature of their activities makes third-party data gathering a challenge at best. Absent an empirical foundation, defenders are often left to speculate as to how successful spam campaigns are and to what degree they are profitable. For example, IBM’s Joshua Corman was widely quoted as claiming that spam sent by the Storm worm alone was generating “millions and millions of dollars every day.”¹ While this claim could in fact be true, we are unaware of any public data or methodology capable of confirming or refuting it.

The key problem is our limited visibility into the three basic parameters of the spam value proposition: the cost to send spam, offset by the “conversion rate” (probability that an email sent will ultimately yield a “sale”), and the marginal profit per sale. The first and last of these are self-contained and can at least be estimated based on the costs charged by

third-party spam senders and through the pricing and gross margins offered by various Internet marketing “affiliate programs.”^a However, the conversion rate depends fundamentally on group actions—on what hundreds of millions of Internet users do when confronted with a new piece of spam—and is much harder to obtain. While a range of anecdotal numbers exist, we are unaware of any well-documented measurement of the spam conversion rate.^b

In part, this problem is methodological. There are no apparent methods for indirectly measuring spam conversion. Thus, the only obvious way to extract this data is to build an e-commerce site, market it via spam, and then record the number of sales. Moreover, to capture the spammer’s experience with full fidelity, such a study must also mimic their use of illicit botnets for distributing email and proxying user responses. In effect, the best way to measure spam is to be a spammer.

In this paper, we have effectively conducted this study, though *sidestepping* the obvious legal and ethical problems associated with sending spam.^c Critically, our study makes use of an *existing* spamming botnet. By infiltrating the botnet parasitically, we convinced it to modify a subset of the spam it *already* sends, thereby directing any interested recipients to Web sites under our control, rather than those belonging to the spammer. In turn, our Web sites presented “defanged” versions of the spammer’s own sites, with functionality removed that would compromise the victim’s system or receive sensitive personal information such as name, address or credit card information.

Using this methodology, we have documented three spam campaigns comprising over 469 million emails. We identified how much of this spam is successfully delivered,

^a Our cursory investigations suggest that commissions on pharmaceutical affiliate programs tend to hover around 40%–50%, while the *retail* cost for spam delivery has been estimated at under \$80 per million.¹⁴

^b The best known among these anecdotal figures comes from the *Wall Street Journal*’s 2003 investigation of Howard Carmack (a.k.a. the “Buffalo Spammer”), revealing that he obtained a 0.00036 conversion rate on 10 million messages marketing an herbal stimulant.³

^c We conducted our study under the ethical criteria of ensuring *neutral actions* so that users should never be worse off due to our activities, while strictly *reducing harm* for those situations in which user property was at risk.

A previous version of this paper appeared in *Proceedings of the 15th ACM Conference on Computer and Communications Security*, Oct. 2008.

how much is filtered by popular antispam solutions, and, most importantly, how many users “click-through” to the site being advertised (*response rate*) and how many of those progress to a “sale” or “infection” (*conversion rate*).

The remainder of this paper is structured as follows. Section 2 describes the economic basis for spam and reviews prior research in this area. Section 4 describes our experimental methodology for botnet infiltration. Section 5 describes our spam filtering and conversion results, Section 6 analyzes the effects of blacklisting on spam delivery, and Section 7 analyzes the possible influences on spam responses. We synthesize our findings in Section 8 and conclude.

2. BACKGROUND

Direct marketing has a rich history, dating back to the nineteenth century distribution of the first mail-order catalogs. What makes direct marketing so appealing is that one can directly measure its return on investment. For example, the Direct Mail Association reports that direct mail sales campaigns produce a response rate of 2.15% on average.⁴ Meanwhile, rough estimates of direct mail *cost per mille*—the cost to address, produce and deliver materials to a thousand targets—range between \$250 and \$1000. Thus, following these estimates it might cost \$250,000 to send out a million solicitations, which might then produce 21,500 responses. The cost of developing these prospects (roughly \$12 each) can be directly computed and, assuming each prospect completes a sale of an average value, one can balance this revenue directly against the marketing costs to determine the profitability of the campaign. As long as the product of the conversion rate and the marginal profit per sale exceeds the marginal delivery cost, the campaign is profitable.

Given this underlying value proposition, it is not at all surprising that bulk direct email marketing emerged very quickly after email itself. The marginal cost to send an email is tiny and, thus, an email-based campaign can be profitable even when the conversion rate is negligible. Unfortunately, a perverse byproduct of this dynamic is that sending as much spam as possible is likely to maximize profit.⁸

While spam has long been understood to be an economic problem, it is only recently that there has been significant effort in modeling spam economics and understanding the value proposition from the spammer’s point of view. Rarely do spammers talk about financial aspects of their activities themselves, though such accounts do exist.^{10,13} Judge et al. speculate that response rates as low as 0.000001 are sufficient to maintain profitability.¹²

However, the work that is most closely related to our own are the several papers concerning “Stock Spam.”^{5,7,9} Stock spam refers to the practice of sending positive “touts” for a low-volume security in order to manipulate its price and thereby profit on an existing position in the stock. What distinguishes stock spam is that it is monetized through price manipulation and not via a sale. Consequently, it is not necessary to measure the conversion rate to understand profitability. Instead, profitability can be inferred by correlating stock spam message volume with changes in the trading volume and price for the associated stocks.

3. THE STORM BOTNET

The measurements in this paper are carried out using the Storm botnet and its spamming agents. Storm is a peer-to-peer botnet that propagates via spam (usually by directing recipients to download an executable from a Web site).

Storm Hierarchy: There are three primary classes of machines that the Storm botnet uses when sending spam. Worker bots make requests for work and, upon receiving orders, send spam as requested. Proxy bots act as conduits between workers and master servers. Finally, the master servers provide commands to the workers and receive their status reports. In our experience there are a very small number of master servers (typically hosted at so-called “bullet-proof” hosting centers) and these are likely managed by the botmaster directly.

However, the distinction between worker and proxy is one that is determined automatically. When Storm first infects a host it tests if it can be reached externally. If so, then it is eligible to become a proxy. If not, then it becomes a worker. All of the bots we ran as part of our experiment existed as proxy bots, being used by the botmaster to ferry commands between master servers and the worker bots responsible for the actual transmission of spam messages.

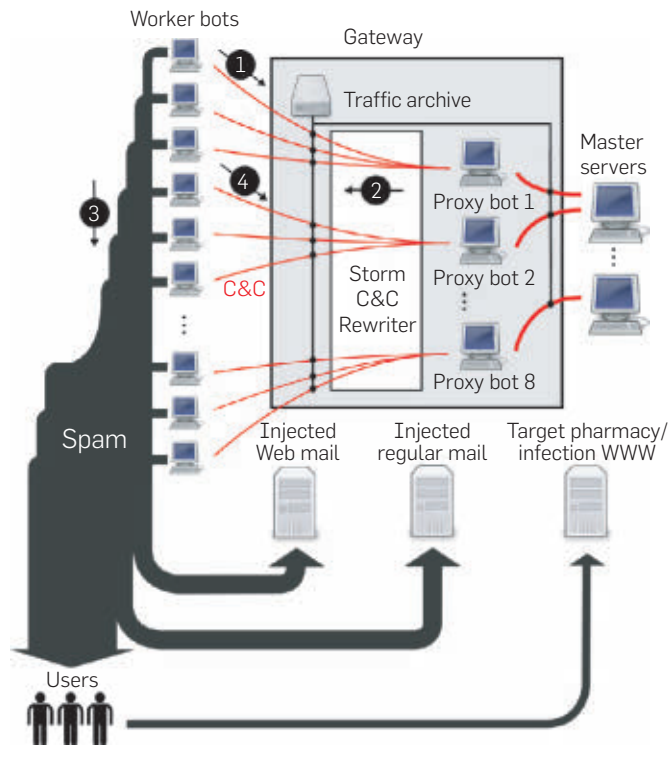
4. METHODOLOGY

Our measurement approach is based on *botnet infiltration*—that is, insinuating ourselves into a botnet’s “command and control” (C&C) network, passively observing the spam-related commands and data it distributes and, where appropriate, actively changing individual elements of these messages in transit. Storm’s architecture lends itself particularly well to infiltration since the proxy bots, *by design*, interpose on the communications between individual worker bots and the master servers who direct them. Moreover, since Storm compromises hosts indiscriminately (normally using malware distributed via social engineering Web sites) it is straightforward to create a proxy bot on demand by infecting a globally reachable host under our control with the Storm malware.

Figure 1 also illustrates our basic measurement infrastructure. At the core, we instantiate eight unmodified Storm proxy bots within a controlled virtual machine environment. The network traffic for these bots is then routed through a centralized gateway, providing a means for blocking unanticipated behaviors (e.g., participation in DDoS attacks) and an interposition point for parsing C&C messages and “rewriting” them as they pass from proxies to workers. Most critically, by carefully rewriting the spam template and dictionary entries sent by master servers, we arrange for worker bots to replace the intended site links in their spam with URLs of our choosing. From this basic capability we synthesize experiments to measure the click-through and conversion rates for several large spam campaigns.

C&C Protocol Rewriting: Our runtime C&C protocol rewriter consists of two components. A custom router redirects potential C&C traffic to a fixed IP address and port, where a user-space proxy server accepts incoming connections and impersonates the proxy bots. This server in turn forwards connections back into the router, which redirects the traffic

Figure 1. The Storm spam campaign dataflow and our measurement and rewriting infrastructure (Section 4). (1) Workers request spam tasks through proxies, (2) proxies forward spam workload responses from master servers, (3) workers send the spam, and (4) return email delivery reports. Our infrastructure infiltrates the C&C channels between workers and proxies.



to the intended proxy bot. Rules for rewriting can be installed independently for templates, dictionaries, and email address target lists. The rewriter logs all C&C traffic between worker and our proxy bots, between the proxy bots and the master servers, and all rewriting actions on the traffic.

Measuring Spam Delivery: To evaluate the effect of spam filtering along the email delivery path to user inboxes, we established a collection of test email accounts and arranged to have Storm worker bots send spam to those accounts. These accounts were created at several different vantage points from which we could evaluate the effectiveness of different email filtering methods. When a worker bot reports success or failure back to the master servers, we remove any success reports for our email addresses to hide our modifications from the botmaster.

We periodically poll each email account (both inbox and “junk/spam” folders) for the messages that it received, and we log them with their timestamps, filtering out any messages not part of this experiment.

Measuring Click-Through and Conversion: To evaluate how often users who receive spam actually visit the sites advertised requires monitoring the advertised sites themselves. Since it is generally impractical to monitor sites not under our control, we have used our *botnet infiltration* method to arrange to have a fraction of Storm’s spam advertise sites of our creation instead.

In particular, we have focused on two types of Storm spam campaigns, a self-propagation campaign designed to spread the Storm malware (typically under the guise of advertising an electronic postcard site) and the other advertising a pharmacy site. These are the two most popular Storm spam campaigns and represent over 40% of recent Storm activity.¹¹ We replaced Storm’s links to its own sites with links to sites under our control, screenshots of which are shown in Figure 2.

These sites have been “defanged” in two important ways: the pharmaceutical site does not accept any personal or payment information, and the self-propagation site advertises a completely benign executable which only phones home to record an execution and exits.

4.1. Measurement ethics

We have been careful to design experiments that we believe are both consistent with current U.S. legal doctrine and are fundamentally ethical as well. While it is beyond the scope of this paper to fully describe the complex legal landscape in which active security measurements operate, we believe the ethical basis for our work is far easier to explain: *we strictly reduce harm*. First, our instrumented proxy bots do not create any new harm. That is, absent our involvement, the same set of users would receive the same set of spam emails sent by the same worker bots. Storm is a large self-organizing system and when a proxy fails its worker bots

Figure 2. Screenshots of the Web sites operated to measure user click-through and conversion.



(a) Pharmaceutical site



(b) Postcard-themed self-propagation site

automatically switch to other idle proxies (indeed, when our proxies fail we see workers quickly switch away). Second, our proxies are passive actors and do not engage themselves in any behavior that is intrinsically objectionable; they do not send spam email, they do not compromise hosts, nor do they even contact worker bots asynchronously. Indeed, their only function is to provide a conduit between worker bots making requests and master servers providing responses. Finally, where we do modify C&C messages in transit, these actions themselves strictly reduce harm. Users who click on spam altered by these changes will be directed to one of our innocuous doppelganger Web sites. Unlike the sites *normally* advertised by Storm, our sites do not infect users with malware and do not collect user credit card information. Thus, no user should receive more spam due to our involvement, but some users will receive spam that is less dangerous than it would otherwise be.

Needless to say, we encourage no one to recreate our experiments without the utmost preparation and care. Interacting with thousands of compromised machines that are sending millions of spam messages is a very delicate procedure, and while we encourage other researchers to build upon our work, we ask that these experiments only be attempted by qualified professionals with no less forethought, legal consultation, or safeguards than those outlined here.

5. EXPERIMENTAL RESULTS

We now present the overall results of our rewriting experiment. We first describe the spam workload observed by our C&C rewriting proxy. We then characterize the effects of filtering on the spam workload along the delivery path from worker bots to user inboxes, as well as the number of users who browse the advertised Web sites and act on the content there.

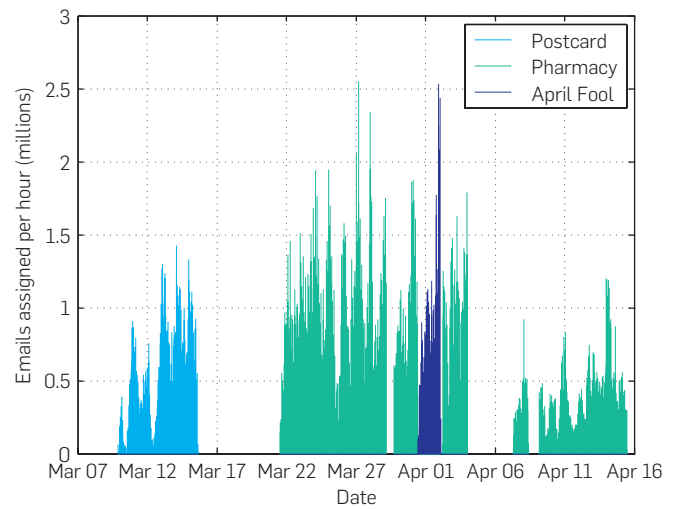
Campaign Datasets: Our study covers three spam campaigns summarized in Table 1. The “Pharmacy” campaign is a 26-day sample (19 active days) of an ongoing Storm campaign advertising an online pharmacy. The “Postcard” and “April Fool” campaigns are two distinct, serial instances of self-propagation campaigns, which attempt to install an executable on the user’s machine under the guise of being postcard software. For each campaign, Figure 3 shows the number of messages per hour assigned to bots for mailing.

Storm’s authors have shown great cunning in exploiting the cultural and social expectations of users—hence the April Fool campaign was rolled out for a limited run around April 1. Our Web site was designed to mimic the earlier

Table 1. Campaigns used in the experiment.

Campaign	Dates	Workers	Emails
Pharmacy	March 21–April 15	31,348	347,590,389
Postcard	March 9–March 15	17,639	83,665,479
April Fool	March 31–April 2	3,678	38,651,124
		Total	469,906,992

Figure 3. Number of email messages assigned per hour for each campaign.

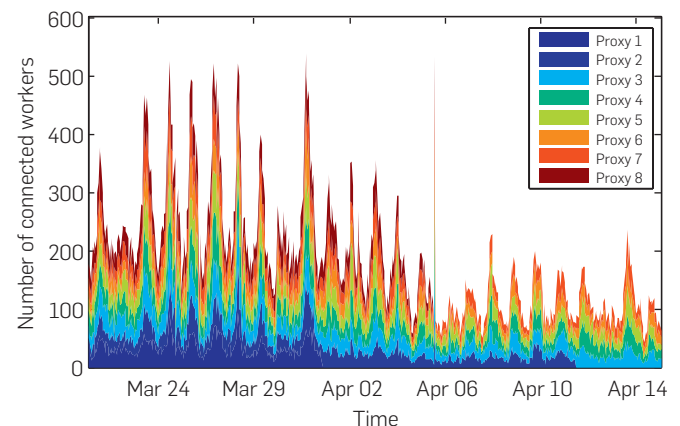


Postcard campaign and thus our data probably does not perfectly reflect user behavior for this campaign, but the two are similar enough in nature that we surmise that any impact is small.

We began the experiment with eight proxy bots, of which seven survived until the end. Figure 4 shows a timeline of the proxy bot workload. The number of workers connected to each proxy is roughly uniform across all proxies (23 worker bots on average), but shows strong spikes corresponding to new self-propagation campaigns. At peak, 539 worker bots were connected to our proxies at the same time.

Most workers only connected to our proxies once: 78% of the workers only connected to our proxies a single time, 92% at most twice, and 99% at most five times. The most prolific worker IP address, a host in an academic network in North Carolina, USA, contacted our proxies 269 times; further inspection identified this as a NAT egress point for 19 individual infections. Conversely, most workers do not connect to more than one proxy: 81% of the workers only connected to a single proxy, 12% to two, 3% to four, 4% connected to five

Figure 4. Timeline of proxy bot workload.



or more, and 90 worker bots connected to all of our proxies. On average, worker bots remained connected for 40 min, although over 40% workers connected for less than a minute. The longest connection lasted almost 81 h.

The workers were instructed to send postcard spam to 83,665,479 addresses, of which 74,901,820 (89.53%) are unique. The April Fool campaign targeted 38,651,124 addresses, of which 36,909,792 (95.49%) are unique. Pharmacy spam targeted 347,590,389 addresses, of which 213,761,147 (61.50%) are unique.

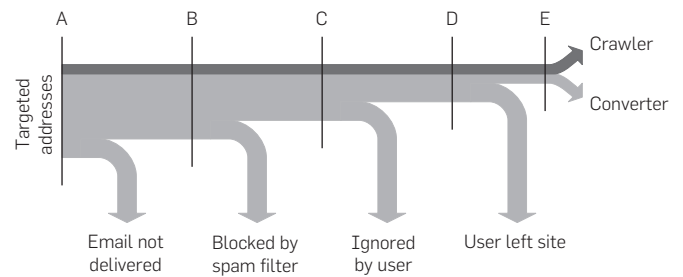
Spam Conversion Pipeline: Conceptually, we break down spam conversion into a pipeline with five “filtering” stages Figure 5 illustrates this pipeline and shows the type of filtering at each stage. The pipeline starts with delivery lists of target email addresses sent to worker bots (Stage A). For a wide range of reasons, workers will successfully deliver only a subset of their messages to an MTA (Stage B). At this point, spam filters at the site correctly identify many messages as spam, and drop them or place them aside in a spam folder. The remaining messages have survived the gauntlet and appear in a user’s inbox as valid messages (Stage C). Users may delete or otherwise ignore them, but some users will act on the spam, click on the URL in the message, and visit the advertised site (Stage D). These users may browse the site, but only a fraction “convert” on the spam (Stage E) by attempting to purchase products (pharmacy) or by downloading and running an executable (self-propagation).

We show the spam flow in two parts, “crawler” and “converter,” to differentiate between real and masquerading users. For example, the delivery lists given to workers contain honeypot email addresses. Workers deliver spam to these honeypots, which then use crawlers to access the sites referenced by the URL in the messages. Since we want to measure the spam conversion rate for actual users, we separate out the effects of automated processes like crawlers, including only clicks we believe to be user-generated in our results.

Table 2 shows the effects of filtering at each stage of the conversion pipeline for both the self-propagation and pharmaceutical campaigns. The number of targeted addresses (A) is simply the total number of addresses on the delivery lists received by the worker bots during the measurement period, excluding the test addresses we injected.

We obtain an estimate of the number of messages delivered to a mail server (B) by relying on delivery reports generated by the workers. The number of messages delivered to a user’s inbox (C) is a much harder value to estimate. We do

Figure 5. The spam conversion pipeline.



not know what spam filtering, if any, is used by each mail provider, and then by each user individually, and therefore cannot reasonably estimate this number in total. It is possible, however, to determine this number for individual mail providers or spam filters. The three mail providers and the spam filtering appliance we used in this experiment had a method for separating delivered mails into “junk” and inbox categories. Table 3 gives the number of messages delivered a user’s inbox for the free email providers, which together accounted for about 16.5% of addresses targeted by Storm (Table 3), as well as our department’s commercial spam filtering appliance. It is important to note that these are results from one spam campaign over a short period of time and should not be used as measures of the relative effectiveness for each service. That said, we observe that the popular Web mail providers all do a very good job at filtering the campaigns we observed, although it is clear they use different methods (e.g., Hotmail rejects most Storm spam at the mail server level, while Gmail accepts a significant fraction only to filter it later as junk).

The number of visits (D) is the number of accesses to our emulated pharmacy and postcard sites, excluding any crawlers. We note that crawler requests came from a small fraction of hosts but accounted for the majority of all requests to our sites. For the pharmacy site, for instance, of the 11,720 unique IP addresses seen accessing the site with a valid unique identifier, only 10.2% were blacklisted as crawlers. In contrast, 55.3% of all unique identifiers used in requests originated from these crawlers. For all nonimage requests made, 87.43% were made by blacklisted IP addresses.

The number of conversions (E) is the number of visits to the purchase page of the pharmacy site, or the number of executions of the fake self-propagation program.

Table 2. Filtering at each stage of the spam conversion pipeline for the self-propagation and pharmacy campaigns. Percentages refer to the conversion rate relative to Stage A.

Stage	Pharmacy		Postcard		April Fool	
A—Spam Targets	347,590,389	100%	83,655,479	100%	40,135,487	100%
B—MTA delivery(est.)	82,700,000	23.8%	21,100,000	25.2%	10,100,000	25.2%
C—Inbox delivery	-	-	-	-	-	-
D—User site visits	10,522	0.00303%	3,827	0.00457%	2,721	0.00680%
E—User conversions	28	0.0000081%	316	0.000378%	225	0.000561%

Table 3. Number of messages delivered to a user's inbox as a fraction of those injected for test accounts at free email providers and a commercial spam filtering appliance. The test account for the Barracuda appliance was not included in the Postcard campaign.

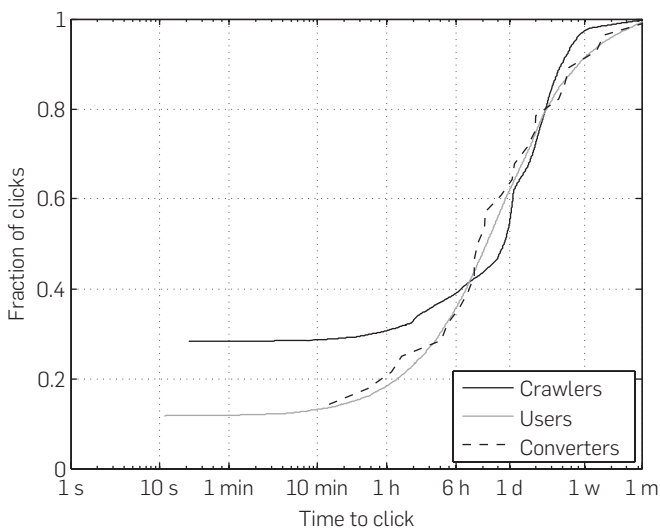
Spam Filter	Pharmacy	Postcard	April Fool
Gmail	0.00683%	0.00176%	0.00226%
Yahoo	0.00173%	0.000542%	None
Hotmail	None	None	None
Barracuda	0.131%	N/A	0.00826%

Our results for Storm spam campaigns show that the spam conversion rate is quite low. For example, out of 350 million pharmacy campaign emails only 28 conversions resulted (and no crawler ever completed a purchase so errors in crawler filtering plays no role). However, a very low conversion rate does not necessary imply low revenue or profitability. We discuss the implications of the conversion rate on the spam conversion proposition further in Section 8.

Time-to-Click: The conversion pipeline shows what fraction of spam ultimately resulted in visits to the advertised sites. However, it does not reflect the latency between when the spam was sent and when a user clicked on it. The longer it takes users to act, the longer the scam hosting infrastructure will need to remain available to extract revenue from the spam.² Put another way, how long does a spam-advertised site need to be online to collect potential revenue?

Figure 6 shows the cumulative distribution of the “time-to-click” for accesses to the pharmacy site. The time-to-click is the time from when spam is sent (when a proxy forwards a spam workload to a worker bot) to when a user “clicks” on the URL in the spam (when a host first accesses the Web site). The graph shows three distributions for the accesses by all users, the users who visited the purchase page (“converters”), and the automated crawlers (14,716 such accesses).

Figure 6. Time-to-click distributions for accesses to the pharmacy site.



The user and crawler distributions show distinctly different behavior. Almost 30% of the crawler accesses are within 20s of worker bots sending spam. This behavior suggests that these crawlers are configured to scan sites advertised in spam immediately upon delivery. Another 10% of crawler accesses have a time-to-click of 1 day, suggesting crawlers configured to access spam-advertised sites periodically in batches. In contrast, only 10% of the user population accesses spam URLs immediately, and the remaining distribution is smooth without any distinct modes. The distributions for all users and users who “convert” are roughly similar, suggesting little correlation between time-to-click and whether a user visiting a site will convert. While most user visits occur within the first 24 h, 10% of times-to-click are a week to a month, indicating that advertised sites need to be available for long durations to capture full revenue potential.

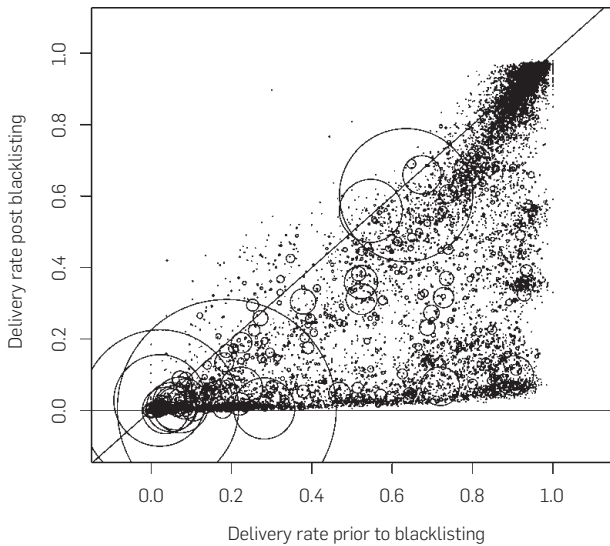
6. EFFECTS OF BLACKLISTING

A major effect on the efficacy of spam delivery is the employment by numerous ISPs of address-based blacklisting to reject email from hosts previously reported as sourcing spam. To assess the impact of blacklisting, during the course of our experiments we monitored the *Composite Blocking List (CBL)*,⁶ a blacklist source used by the operators of some of our institutions. At any given time the CBL lists on the order of 4–6 million IP addresses that have sent email to various spamtraps. We were able to monitor the CBL from March 21–April 2, 2008, from the start of the pharmacy campaign until the end of the April Fool campaign.

We downloaded the current CBL blacklist every half hour, enabling us to determine which worker bots in our measurements were present on the list and how their arrival on the list related to their botnet activity. Of 40,864 workers that sent delivery reports, fully 81% appeared on the CBL. Of those appearing at some point on the list, 77% were on the list prior to our observing their receipt of spamming directives, appearing first on the list 4.4 days (median) earlier. Of those not initially listed but then listed subsequently, the median interval until listing was 1.5 h, strongly suggesting that the spamming activity we observed them being instructed to conduct quickly led to their detection and blacklisting. Of hosts never appearing on the list, more than 75% never reported successful delivery of spam, indicating that the reason for their lack of listing was simply their inability to effectively annoy anyone.

We would expect that the impact of blacklisting on spam delivery strongly depends on the domain targeted in a given email, since some domains incorporate blacklist feeds such as the CBL into their mailer operations and others do not. To explore this effect, Figure 7 plots the per-domain delivery rate: the number of spam emails that workers reported as successfully delivered to the domain divided by number attempted to that domain. The x-axis shows the delivery rate for spams sent by a worker prior to its appearance in the CBL, and the y-axis shows the rate after its appearance in the CBL. We limit the plot to the 10,879 domains to which workers attempted to deliver at least 1,000 spams. We plot

Figure 7. Change in per-domain delivery rates as seen prior to a worker bot appearing in the blacklist (x-axis) vs. after appearing (y-axis). Each circle represents a domain targeted by at least 1,000 analyzable deliveries, with the radius scaled in proportion to the number of delivery attempts.



delivery rates for the two different campaigns as separate circles, though the overall nature of the plot does not change between them. The radius of each plotted circle scales in proportion to the number of delivery attempts, the largest corresponding to domains such as hotmail.com, yahoo.com, and gmail.com.

From the plot we clearly see a range of blacklisting behavior by different domains. Some employ other effective antispam filtering, indicated by their appearance near the origin—spam did not get through even prior to appearing on the CBL blacklist. Some make heavy use of either the CBL or a similar list (y-axis near zero, but x-axis greater than zero), while others appear insensitive to blacklisting (those lying on the diagonal). Since points lie predominantly below the diagonal, we see that either blacklisting or some other effect related to sustained spamming activity (e.g., learning content signatures) diminishes the delivery rate seen at most domains. Delisting followed by relisting may account for some of the spread of points seen here; those few points above the diagonal may simply be due to statistical fluctuations. Finally, the cloud of points to the upper right indicates a large number of domains that are not targeted much individually, but collectively comprise a significant population that appears to employ no effective antispam measures.

7. CONVERSION ANALYSIS

We now turn to a preliminary look at possible factors influencing response to spam. For the present, we confine our analysis to coarse-grained effects.

We start by mapping the geographic distribution of the hosts that “convert” on the spam campaigns we monitored. Figure 8 maps the locations of the 541 hosts that execute the emulated self-propagation program, and the

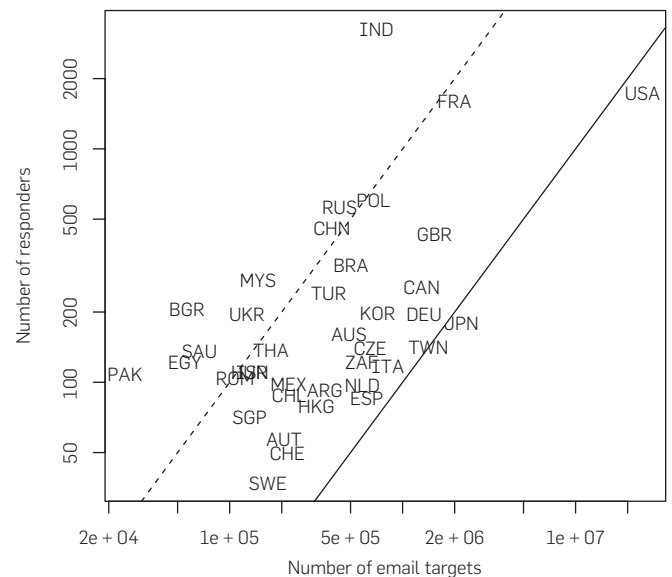
Figure 8. Geographic locations of the hosts that “convert” on spam: the 541 hosts that execute the emulated self-propagation program (light gray), and the 28 hosts that visit the purchase page of the emulated pharmacy site (black).



28 hosts that visit the purchase page of the emulated pharmacy site. The map shows that users around the world respond to spam.

Figure 9 looks at differences in response rates among nations as determined by prevalent country-code email domain TLDs. To allow the inclusion of generic TLDs such as .com, for each email address we consider it a member of the country hosting its mail server; we remove domains that resolve to multiple countries, categorizing them as “international” domains. The x-axis shows the volume of email (log-scaled) targeting a given country, while the y-axis gives the number of responses recorded at Stages A and D in the pipeline (Figure 5), corresponding to Stages A and D in the pipeline (Figure 5). The solid line reflects a response rate of 10^{-4} and the dashed line a rate of 10^{-3} . Not surprisingly, we see that the spam campaigns target email addresses in the United States substantially more than any other

Figure 9. Volume of email targeting (x-axis) vs. responses (y-axis) for the most prominent country-code TLDs. The x and y axes correspond to Stages A and D in the pipeline (Figure 5), respectively.



country. Further, India, France, and the United States dominate responses. In terms of response *rates*, however, India, Pakistan, and Bulgaria have the highest response rates than any other countries (furthest away from the diagonal). The United States, although a dominant target and responder, has the lowest resulting response rate of any country, followed by Japan and Taiwan.

However, the countries with predominant response rates do not appear to reflect a heightened interest in users from those countries in the specific spam offerings. Figure 10 plots the rates for the most prominent countries responding to self-propagation vs. pharmacy spams. The median ratio between these two rates is 0.38 (diagonal line). We see that India and Pakistan in fact exhibit almost exactly this ratio (upper-right corner), and Bulgaria is not far from it. Indeed, only a few TLDs exhibit significantly different ratios, including the United States and France, the two countries other than India with a high number of responders; users in the United States respond to the self-propagation spam substantially more than pharmaceutical spam and vice versa with users in France. These results suggest that, for the most part, per-country differences in response rate are due to *structural* causes (quality of spam filtering, user education) rather than differing degrees of cultural or national interest in the particular promises or products conveyed by the spam.

8. CONCLUSION

This paper describes what we believe is the first large-scale quantitative study of spam conversion. We developed a methodology that uses botnet infiltration to indirectly instrument spam emails such that user clicks on these messages are taken to replica Web sites under our control. Using this methodology we instrumented almost 500 million spam messages, comprising three major campaigns, and quantitatively

characterized both the delivery process and the conversion rate.

We would be the first to admit that these results represent a single data point and are not necessarily representative of spam as a whole. Different campaigns, using different tactics and marketing different products will undoubtedly produce different outcomes. Indeed, we caution *strongly* against researchers using the conversion rates we have measured for these Storm-based campaigns to justify assumptions in any other context. At the same time, it is tempting to speculate on what the numbers we have measured might *mean*. We succumb to this temptation below, with the understanding that few of our speculations can be empirically validated at this time.

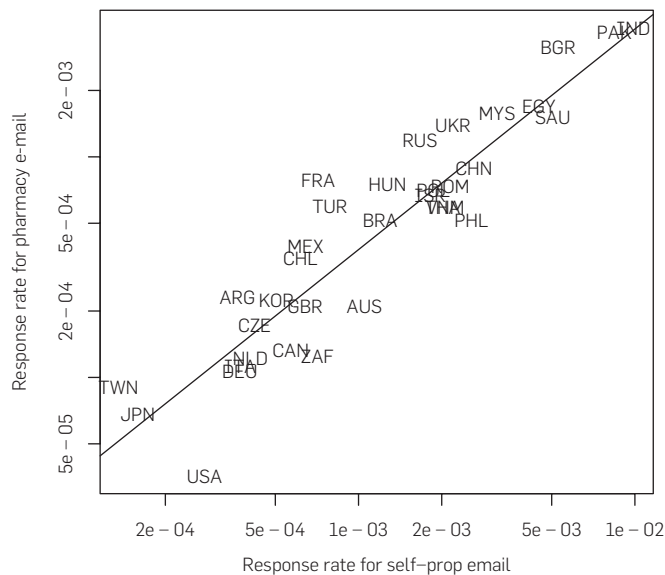
After 26 days, and almost 350 million email messages, only 28 sales resulted—a conversion rate of well under 0.00001%. Of these, all but one was for male-enhancement products and the average purchase price was close to \$100. Taken together, these conversions would have resulted in revenues of \$2,731.88—a bit over \$100 a day for the measurement period or \$140 per day for periods when the campaign was active. However, our study interposed on only a small fraction of the overall Storm network—we estimate roughly 1.5% based on the fraction of worker bots we proxy. Thus, the total daily revenue attributable to Storm’s pharmacy campaign is likely closer to \$7000 (or \$9500 during periods of campaign activity). By the same logic, we estimate that Storm self-propagation campaigns can produce between 3500 and 8500 new bots per day.

Under the assumption that our measurements are representative over time (an admittedly dangerous assumption when dealing with such small samples), we can extrapolate that, were it sent continuously at the same rate, Storm-generated pharmaceutical spam would produce roughly 3.5 million dollars of revenue in a year. This number could be even higher if spam-advertised pharmacies experience repeat business, a bit less than “millions of dollars every day,” but certainly a healthy enterprise.

The next obvious question is, “How much of this revenue is profit?” Here things are even murkier. First, we must consider how much of the gross revenue is actually recovered on a sale. Assuming the pharmacy campaign drives traffic to an affiliate program (and there are very strong anecdotal reasons to believe this is so) then the gross revenue is likely split between the affiliate and the program (an annual net revenue of \$1.75 million using our previous estimate). Next, we must subtract business costs. These include a number of incidental expenses (domain registration, bullet-proof hosting fees, etc.) that are basically fixed sunk costs, and the cost to distribute the spam itself.

Anecdotal reports place the *retail* price of spam delivery at a bit under \$80 per million.¹⁴ In an examination we conducted of some spam-for-hire service advertisements, we found prices ranging from \$70 to over \$100 per million for delivery to US addresses, with substantial discounts available for large volumes. This cost is an order of magnitude less than what legitimate commercial mailers charge, but is still a significant overhead; sending 350M emails would cost more than \$25,000. Indeed, given the net revenues we

Figure 10. Response rates (stage D in the pipeline) by TLD for executable download (x-axis) vs. pharmacy visits (y-axis).



estimate, retail spam delivery would only make sense if it were 20 times cheaper still.

And yet, Storm continues to distribute pharmacy spam—suggesting that it is in fact profitable. One explanation is that Storm’s masters are vertically integrated and the purveyors of Storm’s pharmacy spam are none other than the operators of Storm itself (i.e., that Storm does not deliver these spams for a third-part in exchange for a fee). There is some evidence for this, since the distribution of target email domain names between the self-propagation and pharmacy campaigns is virtually identical. Since the self-propagation campaigns fundamentally must be run by the botnet’s owners, this suggests the purveyor of the pharmacy spam is one and the same. A similar observation can be made in the harvesting of email addresses from the local hard drives of Storm hosts. These email addresses subsequently appear in the target address lists of the pharmacy campaign and self-propagation campaigns alike. Moreover, neither of these behaviors is found in any of the other (smaller) campaigns distributed by Storm (suggesting that these may in fact be fee-for-service distribution arrangements). If true, then the cost of distribution is largely that of the labor used in the development and maintenance of the botnet software itself. While we are unable to provide any meaningful estimates of this cost (since we do not know which labor market Storm is developed in), we surmise that it is roughly the cost of two or three good programmers.

If true, this hypothesis is heartening since it suggests that the third-party *retail* market for spam distribution has not grown large or efficient enough to produce competitive pricing and thus, that profitable spam campaigns require organizations that can assemble complete “soup-to-nuts” teams. Put another way, the profit margin for spam (at least for this one pharmacy campaign) may be meager enough that spammers must be sensitive to the details of how their campaigns are run and are economically susceptible to new defenses.

Acknowledgments

This was one of the most complex measurement studies our group has ever conducted and would have been impossible without the contributions of a large and supportive cast. Here we offer our thanks for their insightful feedback and individual contributions to our effort.

Jordan Hayes provided decidedly nontrivial help with site domain registration. Peter Blair, Paul Karkas, Jamie Knight, and Garrick Lau at Tucows supported this activity (once we convinced them we were not spammers) and allowed us to use reputable registrars. Randy Bush provided overall guidance and help concerning Internet operations and policy issues while Erin Kenneally advised us on legal issues. Brian Kantor set up and managed our DNS, Web, and SMTP servers, while Scott Campbell and Stephen Chan performed massive DNS lookups for us. Jef Poskanzer provided data access for debugging our experiment, Stephen Chenette provided technical assistance and Fallon Chen was our in-house graphic designer. Bill Young and Gregory Ruiz-Ade

set up target email accounts in UCSD’s CSE department. Special thanks to Gabriel Lawrence and Jim Madden of UCSD’s ACT for supporting this activity on UCSD’s systems and networks. Finally, our thanks to the anonymous reviewers for their time and commentary.

This work was made possible by the National Science Foundation grants NSF-0433702 and NSF-0433668 and by generous research, operational and in-kind support from Cisco, Microsoft, HP, Intel, VMWare, ESnet, the Lawrence Berkeley National Laboratory, and UCSD’s Center for Networked Systems. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors or originators and do not necessarily reflect the views of these organizations. ■

References

1. Akass, C. Storm worm ‘making millions a day.’ <http://www.pcv.co.uk/personal-computer-world/news/2209293/strom-worm-making-millions-day>, February 2008.
2. Anderson, D.S., Fleizach, C., Savage, S., Voelker, G.M. Spamscatter: Characterizing internet scam hosting infrastructure. In *Proceedings of the USENIX Security Symposium* (Boston, MA, August 2007).
3. Angwin, J. Elusive Spammer Sends EarthLink on Long Chase. http://online.wsj.com/article_email/SB105225593382372600.html, May 2003.
4. D. M. Association. DMA Releases 5th Annual ‘Response Rate Trends Report.’ <http://www.the-dma.org/cgi/disppressrelease?article=1008>, October 2007.
5. Boehme, R., Ho, T. The effect of stock spam on financial markets. In *Proceedings of the 5th Workshop on the Economics of Information Security (WEIS)* (June 2006).
6. Composite Blocking List (CBL). <http://cbl.abuseat.org/>, March 2008.
7. Frieder, L., Zittrain, J. Spam works: evidence from stock touts and corresponding market activity. *Berkman Center Research Publication*, 2006.
8. Goodman, J., Rounthwaite, R. Stopping outgoing spam. *Proceedings of the 5th ACM Conference on Electronic Commerce* (2004), 30–39.
9. Hanke, M., Hauser, F. On the effects of stock spam emails. *J. Financ. Mark.* *11*, 1 (2008), 57–83.
10. Kirk, J. Former Spammer: ‘I Know I’m Going to Hell.’ <http://www.macworld.com/article/58997/2007/07/spammer.html>, July 2007.
11. Kreibich, C., Kanich, C., Levchenko, K., Enright, B., Voelker, G.M., Paxson, V., Savage, S. On the Spam Campaign Trail. In *First USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET’08)*, April 2008.
12. Judge, W.Y.P., Alperovitch, D. Understanding and Reversing the Profit Model of Spam. In *Workshop on Economics of Information Security 2005 (WEIS 2005)* (Boston, MA, USA, June 2005).
13. Watson, D. All Spammers Go to Hell (posting to funsec list). <http://www.mail-archive.com/funsec%40linuxbox.org/msg03346.html>, July 2007.
14. Wilson, T. Competition May Be Driving Surge in Botnets, Spam. http://www.darkreading.com/document.asp?doc_id=142690, 2008.

Chris Kanich, Kirill Levchenko, Brandon Enright, Geoffrey M. Voelker, and Stefan Savage ([kanich,klevchen,voelker,savage]@cs.ucsd.edu bmenrigh@ucsd.edu), Department of Computer Science and Engineering University of California, San Diego.

Christian Kreibich and Vern Paxson (christian@icir.org, vern@cs.berkeley.edu), International Computer Science Institute Berkeley.

CAREERS

Denison University Assistant Professor of Mathematics and Computer Science

Denison University invites applications for a tenure track position in Mathematics and Computer Science, to begin in August, 2010.

Candidates must have earned a Ph.D. in Mathematics or Computer Science (or a closely related field) and also have a significant background in the other discipline. We seek an individual who is able to teach a variety of undergraduate courses at all levels in either Mathematics or Computer Science and at least through the intermediate level in the other discipline. The successful candidate will also supervise undergraduate research, maintain a strong scientific research program, and contribute in other ways to the department and the college.

Denison University is a highly selective, private liberal arts college enrolling approximately 2,100 undergraduate students. Denison is located in Granville, Ohio, 25 miles east of Columbus. The Department of Mathematics and Computer Science offers B.A. and B.S. degrees in both fields. For more information about Denison and the department, please see our website at <http://www.denison.edu/academics/departments/mathcs/>

The department is currently home to 4 computer scientists and 6 mathematicians; collaboration between members of the two fields is one of our strongest assets.

To apply, please submit a letter of application, a curriculum vita, transcripts of graduate work, a statement on your teaching philosophy, a statement on your research program, and three letters of recommendation online at <https://employment.denison.edu>. At least one recommendation letter must address your teaching effectiveness or potential.

We will begin reviewing applications on October 15, 2009 and will continue until the position is filled. Denison University is an Affirmative Action/Equal Opportunity Employer. Women and minority candidates are especially encouraged to apply.

Dominican University Assistant Professor of Computer Science

Dominican University, a comprehensive Catholic university located ten miles west of downtown Chicago, invites applications for an anticipated full-time tenure-track faculty position in Computer Science beginning in August 2010 at the rank of Assistant Professor. Preferred candidates will have a doctoral degree in Computer Science or a related field. Candidates must have excellent communication skills and the ability and willingness to teach undergraduate courses at all levels, including the university's Core Curriculum. Candidates in all areas of specialization will be considered. Candidates with expertise in computer graphics/gaming, multimedia development or bio-informatics are encouraged to apply. Previous teaching/industry experience is a plus, but is not essential. Teaching

is the primary responsibility, but other responsibilities include scholarly activity and participating in university life and governance. Applications will be reviewed beginning in October 09 until the position is filled. Salary and benefits are competitive. Send a CV, letter of interest, statement of teaching philosophy, three letters of recommendation, and prior teaching evaluations if available. Applications can be sent via email to hr@dom.edu or through the postal service to the following address:

Dominican University
Attn: HR
7900 W Division Street
River Forest, IL 60305

Dominican University is an equal employment opportunity employer seeking applicants from underrepresented groups.

Kuwait University College of Engineering and Petroleum Kuwait

The Department of Computer Engineering at Kuwait University is seeking qualified applicants for a faculty position in the Networks field at the rank of Full Professor or Associate Professor for the academic year 2009-2010.

Required Qualifications:
Ph.D. degree in Computer Engineering with specialization in computer networks from a reputable university. Applicants should have a minimum GPA of 3.0/4.0 or equivalent at the undergraduate level. Applicants should have well-established research experience and publications in refereed international journals. Applicants should have demonstrated outstanding teaching experience in the specified field. The successful candidate is expected to teach at both the undergraduate and graduate levels and to establish an active collaborative research program.

The department has state-of-the-art teaching and research laboratories in various areas supporting the academic programs. Extensive computing network facilities are available for teaching and research. Research is encouraged and funds are available from Kuwait University and other government and private institutions.

To apply send by mail or fax, within six weeks from the date of announcement, a complete application form, with required documents as stated in the application form, a copy of the passport and three recommendation letters, to the following address:

Administration for Academic Staff
Academic Staff Department
University of Kuwait
P.O. Box 5969
Safat 13060
State of Kuwait
Tel: 00965-24844189; Fax: 00965-24849562

For application forms & further information inquiries refer to website:
<http://www.kuniv.edu/forms.php>

Lafayette College Assistant Professor, Tenure Track

Lafayette College Department of Computer Science invites applicants for a tenure-track assistant professor position starting in the fall of 2010. Lafayette College is a selective private liberal arts institution with an undergraduate body of 2300 students. Lafayette College is located in Easton, PA, in the Lehigh Valley, within 80 driving miles of both New York City and Philadelphia.

For more information, visit <http://www.cs.lafayette.edu/search2009.html>

Lafayette College is an Equal Employment Opportunity employer and encourages applications from women and minorities.

University of Pennsylvania Faculty

The University of Pennsylvania invites applicants for tenure-track appointments in computer science to start July 1, 2010. Tenured appointments will also be considered.

The Department of Computer and Information Science seeks individuals with exceptional promise for, or a proven record of, research achievement who will excel in teaching undergraduate and graduate courses and take a position of international leadership in defining their field of study. While exceptional candidates in all areas of core computer science may apply, of particular interest this year are candidates in who are working on the foundations of Market and Social Systems Engineering - the formalization, analysis, optimization, and realization of systems that increasingly integrate engineering, computational, and economic systems and methods. Candidates should have a vision and interest in defining the research and educational frontiers of this rapidly growing field.

The University of Pennsylvania is an Equal Opportunity/Affirmative Action Employer. The Penn CIS Faculty is sensitive to "two-body problems" and would be pleased to assist with opportunities in the Philadelphia region.

For more detailed information regarding this position and application link please visit:

<http://www.cis.upenn.edu/departamental/facultyRecruiting.shtml>

University of Pennsylvania Lecturer

The University of Pennsylvania invites applicants for the position of Lecturer in Computer Science to start July 1, 2010. Applicants should hold a graduate degree (preferably a Ph.D.) in Computer Science or Computer Engineering, and have a strong interest in teaching with practical application. Lecturer duties include undergraduate and graduate level courses within the

Master of Computer and Information Technology program (www.cis.upenn.edu/grad/mcit/). Of particular interest are applicants with expertise and/or interest in teaching computer hardware and architecture. The position is for one year and is renewable annually up to three years. Successful applicants will find Penn to be a stimulating environment conducive to professional growth in both teaching and research.

The University of Pennsylvania is an Equal Opportunity/Affirmative Action Employer. The Penn CIS Faculty is sensitive to "two-body problems" and would be pleased to assist with opportunities in the Philadelphia region.

For more detailed information regarding this position and application link please visit:

<http://www.cis.upenn.edu/departamental/facultyRecruiting.shtml>

The University of Texas at Tyler Computer Science Faculty Position

The Department of Computer Science invites applications for a tenure-track faculty position at the assistant professor level. An earned doctorate in computer science, demonstrated English communication skills, and commitment to excellence in teaching, research, scholarship and service are required. The successful candidate must demonstrate a potential for externally funded research. While all areas of specialization will be considered, preferred specialties include computer security, bioinformatics, high-performance computing, and information systems. UT Tyler,

a component of The University of Texas System, is located in the beautiful East Texas lake country on the I-20 corridor, about 100 miles east of Dallas and 200 miles north of Houston. With nine full-time faculty members and two staff assistants, the department offers baccalaureate and graduate degree programs in a quality learning environment with new teaching classrooms and research labs in the Ratliff Engineering and Science Complex. Further information about the department, college, university and the Tyler area can be found by visiting the UT Tyler web site at <http://www.uttyler.edu>. Interested individuals should send a letter of application, curriculum vitae, and the names and contact information of three references to: Faculty Search Committee, Department of Computer Science, The University of Texas at Tyler, 3900 University Boulevard, Tyler, TX 75799 or via email to cssearch@uttyler.edu. The search committee will begin reviewing applications in October 2009 and will continue until the position is filled. The University of Texas at Tyler is an Equal Opportunity Employer. Women and minorities are strongly encouraged to apply. The successful applicant must be able to demonstrate eligibility to work in the United States.

Washington State University Vancouver Faculty Position in Computer Science

FACULTY POSITION IN COMPUTER SCIENCE – Washington State University Vancouver invites applications for a tenure-track position at the assistant professor level beginning 8/16/2010 in the School of Engineering and Computer Science. Candidates are sought with expertise in computer security, **large scale data management, data mining or cyber-physical systems.**

Required qualifications: Ph.D. in Computer Sci-

ence or Computer Engineering at the time of employment and demonstrated ability to (1) develop a funded research program, (2) establish strong industrial collaborations, and (3) teach undergraduate and graduate courses and laboratories. Preferred qualifications: relevant industry experience, experience with ABET accreditation, and commitment to working with diverse student and community populations. WSU Vancouver is committed to building a culturally diverse educational environment.

WSU Vancouver serves about 2600 graduate and undergraduate students and is **fifteen miles north of Portland, Oregon.** The School of Engineering and Computer Science (ENCS) offers ABET-accredited BS and MS degrees in mechanical engineering and computer science. The State recently authorized a new BS-EE program along with funding for a second **new building for the ENCS.** The rapidly growing ENCS equally values both research and teaching. WSU is Washington's land grant university with faculty and programs on four campuses. For more information: <http://www.vancouver.wsu.edu/encs>.

Applications must include: (1) cover letter with a clear description of experience relevant to the position; (2) vita including a list of references; and (3) **maximum three-page total** summary statement of research and teaching experience. This statement must describe how the candidate's research activity will expand or complement the current research in ENCS. It must also list the existing ENCS courses the candidate can teach and proposed new courses the candidate can develop. Application deadline is December 4, 2009. All materials should be mailed to CS Search Committee, School of ENCS - VELS 130, Washington State University, 14204 NE Salmon Creek Avenue, Vancouver, WA 98686-9600. WSU employs only US citizens and lawfully authorized non-citizens. WSU is an EO/AA educator and employer.



Announcement of an open position at the
Faculty of Informatics,
Vienna University of Technology, Austria

Full Professor (tenured) in

Algorithms and Data Structures

The successful candidate is expected to lead his/her own group and to conduct research and teaching in the area of Algorithms and Data Structure. Ideally, candidates are sought, who are able to combine theoretical research with novel applications.

Applicants are expected to have an outstanding academic record and experience with research projects as well as in university teaching.

We offer excellent working conditions in an attractive research environment in a town with an exceptional quality of living.

A more detailed announcement and information on how to apply can be found at <http://www.informatik.tuwien.ac.at/ADS.pdf>

Application deadline: **November 20, 2009**



ADVERTISING IN CAREER OPPORTUNITIES

How to Submit a Classified Line Ad: Send an e-mail to acmm mediasales@acm.org. Please include text, and indicate the issue/or issues where the ad will appear, and a contact name and number.

Estimates: An insertion order will then be e-mailed back to you. The ad will be typeset according to CACM guidelines. NO PROOFS can be sent. Classified line ads are NOT commissionable.

Rates: \$325.00 for six lines of text, 40 characters per line. \$32.50 for each additional line after the first six. The MINIMUM is six lines.

Deadlines: 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 212-626-0654 or
acmm mediasales@acm.org**



Peter Winkler

DOI:10.1145/1562164.1562191

Puzzled Solutions and Sources

Last month (August 2009, p. 104) we posted a trio of brainteasers, including one as yet unsolved, concerning probability and intuition.

1. Last Night in Vegas

Solution. This puzzle was passed to me by math-puzzle connoisseur Elwyn Berlekamp during the seventh Gathering for Gardner (<http://www.g4g4.com/>), March 2006, in Atlanta, GA, one of an ongoing series of meetings dedicated to the great puzzle proselytizer Martin Gardner. It later appeared in Berlekamp's and Joe Buhler's "Puzzles Column" in *The Emissary* newsletter (<http://www.msri.org/communications/emissary/index.html>) published by the Mathematical Sciences Research Institute, Berkeley, CA.

The idea is that most people have pretty good intuition concerning the "law of large numbers," which says roughly that repeated random events, like betting on roulette, tend in the long run to produce approximately the result predicted by probabilities. At a Las Vegas roulette table in the puzzle, each single-number bet loses an average of $\$1 - 1/38 \times \$36 = \$1/19$, or about five cents. Thus, a run of 105 bets loses an average of \$5.53 in total, even if it is your birthday. Sounds like your probability of coming out ahead should be small.

However, averages don't tell the whole story, as we are reminded by the legend of the statistician who drowned in a river of average depth three inches. As it turns out, 105 bets are not nearly enough to invoke the law of large numbers. Much of the time (exact probability is $(105 \times 104 \times 103 / 3 \times 2 \times 1) \times (1/38)^3 \times (37/38)^{102}$, or around 0.225) you will win exactly three times, putting you ahead by a hair. You would then have \$108 for your \$105 investment. A few

more calculations, and you'll find that the probability of coming out ahead is about .5254, or more than a half.

This doesn't mean you have Las Vegas by the throat. Failing to get your three wins, you'd lose a lot of your money, on average, paying \$5.53 for your roulette adventure.

For a more extreme example of this phenomenon, suppose you approach the roulette table with \$255 but need \$256 to pay your registration fee for an ACM conference at the same hotel. Your best course would be to plan on betting \$1, then \$2, then \$4, \$8, \$16, \$32, \$64, and finally \$128 on red (or black). The first time you win, you collect double your stake and quit immediately, now with exactly the \$256 you need. You fail only if you lose all eight bets (and all your money). But failure occurs with probability only $(20/38)^8$, or less than .006, so you'd get to attend the conference more than 99.4% of the time.

You could also then quit gambling for the rest of your life. Highly recommended.

2. Fully Booked Aircraft

Solution. I heard this puzzle at the fifth Gathering for Gardner, from Ander Holroyd of Microsoft Research. It seems to have been circulating for years, though it often happens that people who have heard it before are mystified the second time around as well. The key is that the last empty seat must have been the one assigned to either the last passenger or to the first. However, just because we have only two cases doesn't mean the prob-

abilities are necessarily equal, as victims of the Monte-Hall-and-the-three-doors puzzle can testify.

Here, however, the two seats play identical roles in the boarding process; each passenger is as likely to take one seat as the other, as long as both are free. Hence, the probability that it is indeed his/her own seat that the last passenger finds open is exactly $1/2$. The argument works with 100 replaced by any number greater than one.

3. Random Arcade

Conjecture. My intuition, and perhaps yours, too, suggests that the best possible situation is if each gumball machine disgorges $n+1$ gumballs with probability $1/(n+1)$, otherwise none. That way, the player putting a coin in each machine would succeed as long as at least one of the n machines pays off. What is the player's success probability in this scenario?

Failure requires that each machine refuses to cooperate, which happens with probability $(1 - 1/(n+1))^n$. The player succeeds with probability one minus that expression. For n equals one through six, the formula gives success probabilities $1/2$, $5/9$, $37/64$, $369/625$, $4,651/7,776$, and $70,993/117,649$; to the nearest thousandth, it would be .500, .556, .578, .590, .598, and .603. The numbers approach $1 - 1/e \sim .632$, from below, as n increases. Thus, the answer appears to be $1 - 1/e$.

However, no one has managed to prove that you can never do better than $1 - 1/e$. The puzzle's creator, Uriel Feige of the Weizmann Institute of Science, Rehovot, Israel, has shown that the success probability can never exceed $12/13 \sim .923$. Can you get a better bound?

All readers are encouraged to submit prospective puzzles for future columns to puzzled@cacm.acm.org.

Peter Winkler (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.

LAST BYTE

[CONTINUED FROM P. 112] advertising. The bifurcation of humanity could be sustained only so long as those on the receiving end have money to spend. But as more things become free in order to support advertising, fewer of us will be making money. The dénouement would probably be some sort of violent swing toward socialism.

This might sound like an extreme scenario, but consider how much more difficult it is for certain creative people to earn a living today than they did before the public Internet became a global social phenomenon. The most tormented examples are probably recording musicians and investigative journalists.

Alas, it is now common to hear suggestions that people in this predicament should revert to retro (inevitably more physical) strategies of sustenance, like selling branded T-shirts and other merchandise. This is a sad reversal of what had been one of the brightest aspects of technological progress. Prior to the centrality of “open culture” and the rise of online collectivization, technological progress generally supported ever more cerebral, creative, and comfortable means of making a living.

Now extrapolate: How long will it be before cheap fabricating robots are able to download T-shirt designs from the cloud and automatically manufacture customized clothing as easily as one downloads music today? And how long after that will it be before personal robots are able to build copies of the latest medical implant or other gadgets from an online design? The answers are likely to be measured in decades, not centuries. If robotics is eventually good enough to harvest the garbage dumps of the world for materials and transform them into manufactured products, then a plateau will have been reached. At that time, all consumer technology will become media technology. Even those who hoped to make a living from T-shirts will join the investigative journalist and recording musician in poverty.

How far back in history toward the stone age will people have to devolve in order to find a way to make a living

How long will it be before cheap fabricating robots are able to download T-shirt designs from the cloud and automatically manufacture customized clothing as easily as one downloads music today?

when fabricating robots are that good? Will people be forced by the marketplace to work the fields, as academics did under various Maoist-type regimes? Not with good robots around. Surely, robots will eventually also do a better job tending the crops.

If you go back to some of the earliest thinking about how information technology might interact with the patterns of human life, you'll find examples of people who thought ahead to this potential dilemma. For instance, Ted Nelson, probably the first person to really think through how something like the Web might be built and how it would influence human society, proposed in the 1960s a design in which each copy of a file existed, from a logical point of view, in only one instance. Any user could make micropayments to gain access. The conflict between file sharing and DRM would be defused because there would be little motivation to make copies. Accessing files would be enticingly cheap, but everyone would make some incremental amount of money from sharing files with everyone else. A new social contract would emerge based on self-interest. This was not just a proposal to extend capitalism,

but to broaden its benefits to a greater variety of people, since all would be able to upload interesting bits as needed.

A popular objection when Nelson proposed this design was that few people had anything of interest or value to say, and if they tried to say what they could, no one else would be interested. Fortunately, the rise of social networking has proved these objections unfounded.

I directly experienced a later period, in the 1970s and 1980s, when Nelson was no longer a solitary pioneer. Much of the underlying architecture and ideology that guides the public Internet today appeared in rough cut during those years. The ideas had shifted. Nelson was attacked by the campus left of the time over his willingness to imagine a future in which money continued to be important. Meanwhile, the culture of AI fascinated engineers, drawing their attention away from the problem of how to reward human creativity that had so fascinated Nelson.

We ended up with an Internet and Web that is, for the moment, a sort of cross between mass collective implementation of a Turing Test, through designs like Twitter, and the clumsy fantasy of armchair pseudo-Maoists. I realize these words could strike many as alarmist. If this is the case for you, please look into the history of collectivist design in human affairs. Such designs often appear enlightened at first, with a special way of enchanting idealistic young people. But they have also engendered the worst social disasters of the past century.

That's why I reject the idea that a collective or emergent intelligence is appearing through the computing clouds. We'll never know if it's really there, or if we have collectively become idiots. □

Jaron Lanier is a computer scientist interested in interpersonal perception, biomimetic computing, and new displays and sensors. He received a Career Award from the IEEE in 2009 for his lifetime contributions to virtual reality research and is presently working at Microsoft on intriguing unannounced projects.

© 2009 ACM 0001-0782/09/0900 \$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:10.1145/1562164.1562192

Jaron Lanier

Future Tense

Confusions of the Hive Mind

Cherish the individual.

BE CAUTIOUS ABOUT the artificial intelligence approach to computer science. It is impossible to differentiate the actual achievement of AI from the degree to which people change when confronted with what is purported to be intelligent technology. We humans are vulnerable to bending over backward, sometimes making ourselves significantly stupider, in order to make an algorithm seem smart. A great many people in the U.S., as well as elsewhere, demonstrated this danger when they interacted foolishly with deeply flawed algorithms related to the credit and mortgage industries.

There is an even greater economic danger ahead as it relates to the idea of AI. If we are gullible enough to expect emergent large-scale intelligence to arise from the vast connections of the worldwide Internet, as has been proposed with increasing frequency in *Communications* and elsewhere, then we risk undermining the value we place on human labor and creativity. We might thus ruin the most successful design yet invented for the purpose of generating and preserving individual human dignity and liberty—capitalism.

Those who believe in the imminent arrival of global AI (possibly emerging from the computing clouds) pretend that all the information we humans upload actually comes from some mysterious supernatural dimension. There's an economic component to the way we lie to ourselves to support this confusion. Millions of us anonymously upload our online offerings—thoughts, pictures, videos, links, votes,



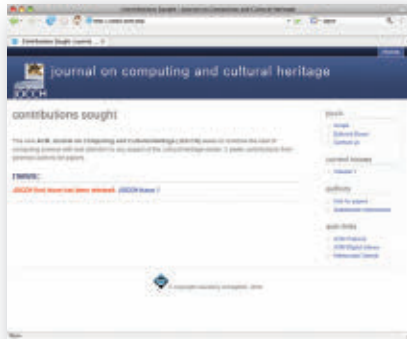
and more. Or, if not anonymously, we often express ourselves in such a fragmentary way, as with tweets, that there is no room left for personality. Under these circumstances we accept that we will not be paid for our acts of expression, as if we are engaged in a massive economic ritual to reify the falsehood that a global supernatural brain is speaking, instead of us.

The idea of creativity emerging autonomously from the computing clouds has the potential to ruin what

might be called the endgame of basic technological development. Will technology good enough to provide comfort and security usher in a golden age for all? Or will we diverge into two species, one relatively lucky, the other relatively left out, as predicted by H.G. Wells in his novel *The Time Machine* in 1898?

The rarified beneficiaries might turn out to be the owners of the computing clouds, while the rest might be inundated with [CONTINUED ON P. 111]

World Class Journals from ACM



ACM Journal on Computing and Cultural Heritage
<http://jocch.acm.org/>

JOCCH publishes papers of significant and lasting value in all areas relating to the use of ICT in support of Cultural Heritage.

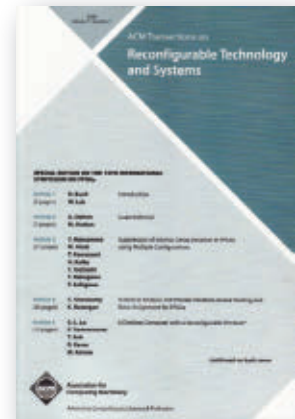
Order Codes: Online – 273
 ISSN: 1556-4673
 Pricing: \$ 50 Professional
 \$ 45 Student
 \$ 150 Non-Member
 \$ 18 Air Service



ACM Transactions on Accessible Computing
<http://www.is.umbc.edu/taccess/>

TACCESS is a quarterly journal that publishes refereed articles addressing issues of computing as it impacts the lives of people with disabilities.

Order Codes: Print – 174 Online – 274
 ISSN: 1936-7228
 Pricing: \$ 50 Professional
 \$ 45 Student
 \$ 150 Non-Member
 \$ 18 Air Service



ACM Transactions on Reconfigurable Technology and Systems
<http://tret.s.acm.org/>

TRETs is a peer-reviewed and archival journal that covers reconfigurable technology, systems, and applications on reconfigurable computers.

Order Codes: Print – 170 Online – 270
 ISSN: 1556-4681
 Pricing: \$ 50 Professional
 \$ 45 Student
 \$ 150 Non-Member
 \$ 18 Air Service

*Air Service is for residents outside North America only.

ACM publishes over 40 magazines and journals that cover a vast array of established as well as emerging areas of the computing field. IT professionals worldwide depend on ACM's publications to keep them abreast of the latest technological developments and industry news in a timely, comprehensive manner of the highest quality and integrity. For a complete listing of ACM's leading magazines & journals, including our renowned *Transaction Series*, please visit the ACM publications homepage at:

www.acm.org/pubs

PLEASE CONTACT ACM MEMBER SERVICES TO PLACE AN ORDER

Phone: 1.800.342.6626 (U.S. and Canada)
 +1.212.626.0500 (Global)
 Fax: +1.212.944.1318
 (Hours: 8:30 AM – 4:30 PM, Eastern Time)
 Email: acmhelp@acm.org
 Mail: ACM Member Services
 General Post Office
 PO Box 30777
 New York, NY 10087-0777 USA



Association for
Computing Machinery

Advancing Computing as a Science & Profession

Emerging Software Technologies


O R L A N D O 2 0 0 9

Seeding the Clouds
The New World of Software Development
Software Engineering Tools:
The Next Generation

Disney's Contemporary
Resort, Orlando, FL
October 25-29 2009

Keynotes & Invited Speakers:

Barbara Liskov (Turing Award), Tom Malone (MIT),
Jeanette Wing (CMU, NSF), Gerard J. Holzman (NASA JPL),
Robert Johnson (Facebook), Jimmy Wales (Wikipedia)



PROGRAM CHAIR
Gary T. Leavens
University of Central Florida
papers@oopsla.org

CONFERENCE CHAIR
Shail Arora, Gradepoint Inc.
chair@oopsla.org

ONWARD! CHAIR
Bernd Brügge
Technische Universität München
chair@onward-conference.org



Association for
Computing Machinery

Advancing Computing as a Science & Profession

*OOPSLA is sponsored by
ACM SIGPLAN in cooperation with SIGSOFT*

www.oopsla.org