

COMMUNICATIONS OF THE ACM

CACM.AC.M.ORG

07/2009 VOL.52 NO.07

Barbara Liskov ACM's A.M. Turing Award Winner

Steps Toward
Self-Aware Networks

The Metropolis Model

Why Computer Science
Doesn't Matter

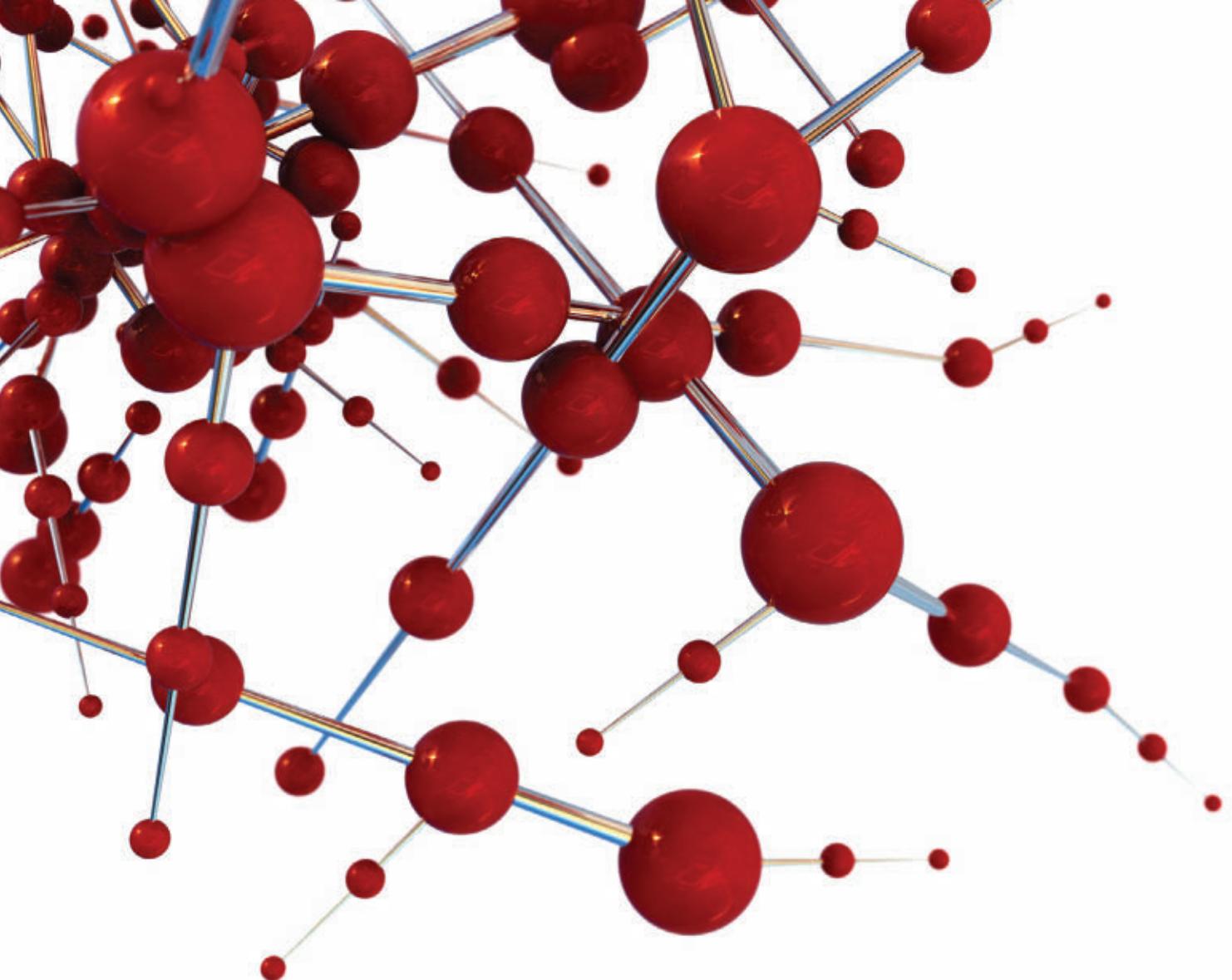
Probabilistic
Databases

The Five-Minute Rule
20 Years Later



Association for
Computing Machinery





**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.

Noteworthy Titles

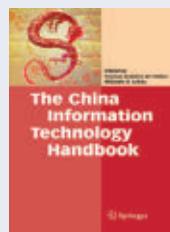


Model Driven Engineering and Ontology Development

D. Gaševic, Simon Fraser University, Surrey, BC, Canada; D. Djuric, V. Devedžić, University of Belgrade, Serbia and Montenegro

Defining a formal domain ontology is considered a useful, not to say necessary step in almost every software project. This is because software deals with ideas rather than with self-evident physical artefacts. However, this development step is hardly ever done, as ontologies rely on well-defined and semantically powerful AI concepts such as description logics or rule-based systems, and most software engineers are largely unfamiliar with these. This book fills this gap by covering the subject of MDA application for ontology development on the Semantic Web. The writing is technical yet clear, and is illustrated with examples. The book is supported by a website.

2nd ed. 2009. 153 illus. Hardcover
ISBN 978-3-642-00281-6 ► \$69.95



The China Information Technology Handbook

P. Ordóñez de Pablos, University of Oviedo, Asturias, Spain; M. D. Lytras, University of Patras, Greece (Eds.)

The book is a reference for those interested in information technologies and emerging management practices in China. This reference is timely, responding to the high demand of society to adopt emerging technologies in all aspects of business and economic activity, towards innovative solutions to research problems and high performance systems. The emphasis on information technologies and management provides a unique value proposition and gives characteristics of flexibility and adoption to diverse audiences. The subject area is a combination of global information technology and management along with strategic management of IT.

2009. XVII, 433 p. 82 illus. Hardcover
ISBN 978-0-387-77742-9 ► \$199.00

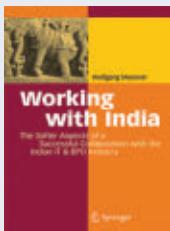


Enterprise Governance and Enterprise Engineering

J. A. Hoogervorst, Sogeti Netherlands BV, Diemen, The Netherlands

Two themes underpin the fundamentally different views outlined in this book. First, the competence-based perspective on governance, whereby employees are viewed as the crucial core for effectively addressing the complex, dynamic and uncertain enterprise reality, as well as for successfully defining and operationalizing strategic choices. Second, enterprise engineering as the formal conceptual framework and methodology for arranging a unified and integrated enterprise design, which is a necessary condition for enterprise success.

2009. XXV, 429 p. 103 illus.
(The Enterprise Engineering Series) Hardcover
ISBN 978-3-540-92670-2 ► \$79.95



Working with India

The Softer Aspects of a Successful Collaboration with the Indian IT & BPO Industry

W. Messner, Capgemini India, Bangalore, India
Globalization requires effective international and cross-cultural collaboration. This book is a start on the journey of cultural appreciation for managers, project leaders, and offshore coordinators working together with Indians. It is also a resource for business managers and company strategists seeking to understand the softer aspects behind the headlines that the Indian IT and BPO industry so frequently creates.

2009. XIII, 178 p. 73 illus. Hardcover
ISBN 978-3-540-89077-5 ► \$59.95



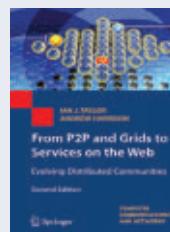
eCulture

Cultural Content in the Digital Age

A. M. Ronchi, Politecnico di Milano, Italy

Do virtual museums really provide added value to end-users, or do they just contribute to the abundance of images? Does the World Wide Web save endangered cultural heritage, or does it foster a society with less variety? These and other related questions are raised and answered in this book, the result of a long path across the digital heritage landscape. It provides a comprehensive view on issues and achievements in digital collections and cultural content.

2009. XXX, 456 p. 271 illus. in color. Hardcover
ISBN 978-3-540-75273-8 ► \$129.00



From P2P and Grids to Services on the Web

Evolving Distributed Communities

I. J. Taylor, A. Harrison, University of Cardiff, UK

This broad-ranging new edition of a classic textbook/reference provides a comprehensive overview of emerging distributed-systems technologies and has been significantly enhanced and extended to cover the many new, state-of-the-art infrastructures and technologies that have since appeared. The focus is also broadened, retaining the technical aspects, but additionally including useful historical contexts for each of the technologies.

2nd ed. 2009. XXIV, 448 p. 123 illus.
(Computer Communications and Networks)
Softcover
ISBN 978-1-84800-122-0 ► \$79.95

Easy Ways to Order for the Americas ► Write: Springer Order Department, PO Box 2485, Secaucus, NJ 07096-2485, USA ► **Call: (toll free)** 1-800-SPRINGER

► **Fax:** 1-201-348-4505 ► **Email:** orders-ny@springer.com or **for outside the Americas ► Write:** Springer Customer Service Center GmbH, Haberstrasse 7,

69126 Heidelberg, Germany ► **Call:** +49 (0) 6221-345-4301 ► **Fax:** +49 (0) 6221-345-4229 ► **Email:** orders-hd-individuals@springer.com

► Prices are subject to change without notice. All prices are net prices.

014244x

COMMUNICATIONS OF THE ACM

Departments

- 5 **Editor's Letter**
Open, Closed, or Clopen Access?
By Moshe Y. Vardi

- 6 **Publisher's Corner**
Communications'
Annual Report Card
By Scott E. Delman

- 8 **Letters To The Editor**
Inspire with Introductory Computer Science

- 10 **blog@CACM**
Sharing Ideas, Writing Apps, and Creating a Professional Web Presence
Greg Linden reveals his new approach to reading research papers, Mark Guzdial discusses how to encourage students to write computer programs, and Tessa Lau shares her ideas about the importance of Web visibility.

- 12 **CACM Online**
Moving Forward and Backward
By David Roman

- 45 **Calendar**

- 116 **Careers**

Last Byte

- 120 **Q&A**
Liskov on Liskov
Barbara Liskov talks about her groundbreaking work in data abstraction and distributed computing.
By Leah Hoffmann

News

- 13 **Contemporary Approaches to Fault Tolerance**

Thanks to computer scientists like Barbara Liskov, researchers are making major progress with cost-efficient fault tolerance for Web-based systems.

By Alex Wright

- 16 **Toward Native Web Execution**

Several software projects are narrowing the performance gap between browser-based applications and their desktop counterparts. In the process, they're creating new ways to improve the security of Web-based computing.

By Kirk L. Kroeker

- 18 **Are We Losing Our Ability to Think Critically?**

Computer technology has enhanced lives in countless ways, but some experts believe it might also be affecting people's ability to really *think*.

By Samuel Greengard

- 20 **Liskov's Creative Joy**

Barbara Liskov muses about the creative process of problem solving, finding the perfect design point, and pursuing a research path.

By Karen A. Frenkel

- 23 **Master of Connections**

Jon Kleinberg is honored for his pioneering research on the Web and social networking.

By Alan Joch

- 25 **ACM Award Winners**

Among this year's distinguished honorees are Barbara Liskov of Massachusetts Institute of Technology and Jon Kleinberg of Cornell University.

Viewpoints

- 28 **Legally Speaking**
The Dead Souls of the Google Book Search Settlement

Why the Google Book Search settlement agreement under consideration could result in an extensive restructuring of the book industry.

By Pamela Samuelson

- 31 **Technology Strategy and Management**
Globalization of Knowledge-Intensive Professional Services

Does the trend toward standardization and modularization of professional services make outsourcing inevitable?

By Mari Sako

- 34 **The Business of Software**
The Cliché Defense

A guide to playing the ploys frequently employed by cliché-driven management.

By Phillip G. Armour

- 37 **Viewpoint**
Why Computer Science Doesn't Matter

Aligning computer science with high school mathematics can help turn it into an essential subject for all students.

By Matthias Felleisen
and Shriram Krishnamurthi

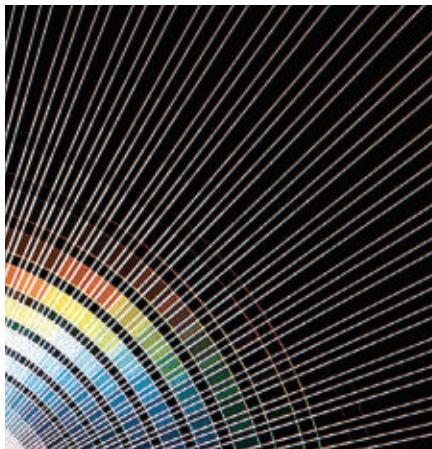
- 41 **Point/Counterpoint**
CS Education in the U.S.: Heading in the Wrong Direction?

Considering the most effective methods for teaching students the fundamental principles of software engineering.

By Robert Dewar and Owen Astrachan



Association for Computing Machinery
Advancing Computing as a Science & Profession

Practice

- 48 The Five-Minute Rule 20 Years Later (and How Flash Memory Changes the Rules)**
Revisiting Gray and Putzolu's famous rule in the age of Flash.
By Goetz Graefe

- 60 Fighting Physics: A Tough Battle**
The laws of physics and the Internet's routing infrastructure affect performance in a big way.
By Jonathan M. Smith

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



About the Cover:
Boston-based photographer Jared Leeds captured Barbara Liskov, recipient of the 2008 ACM A.M. Turing Award, in front of the Stata Center on the MIT campus where she is the Ford Professor of Engineering.

Contributed Articles

- 66 Steps Toward Self-Aware Networks**
Network software adapts to user needs and load variations and failures to provide reliable communications in largely unknown networks.
By Erol Gelenbe
- 76 The Metropolis Model: A New Logic for Development of Crowdsourced Systems**
It takes a city of developers to build a big system that is never done.
By Rick Kazman and Hong-Mei Chen

Review Articles

- 86 Probabilistic Databases: Diamonds in the Dirt**
Treasures abound from hidden facts found in imprecise data sets.
by Nilesh Dalvi, Christopher Ré, and Dan Suciu

Research Highlights

- 96 Technical Perspective**
The Ultimate Pilot Program
By Stuart Russell and Lawrence Saul

- 97 Apprenticeship Learning for Helicopter Control**
By Adam Coates, Pieter Abbeel, and Andrew Y. Ng

- 106 Technical Perspective**
A Compiler's Story
By Greg Morrisett

- 107 Formal Verification of a Realistic Compiler**
By Xavier Leroy

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.

Improving the Cyber Security of SCADA Communication Networks

Sandip C. Patel, Ganesh D. Bhatt, and James H. Graham

Adoption Leadership and Early Planners: Comcast's IP Upgrade Strategy

Anat Hovav and Ciprian Popiviciu

Software Project Scope Alignment: An Outcome-Based Approach

Richard W. Woolridge, David P. Hale, Joanne E. Hale, and R. Shane Sharpe

A Relevancy-Based Services View for Driving Adoption of Wireless Web Services in the U.S.

Arvind Malhotra and Claudia Kubowicz Malhotra

Churchman's Inquirers as Design Templates for Knowledge Management Systems

James L. Parrish, Jr. and James F. Courtney, Jr.

Security Challenges of EPCglobal Network

Benjamin Fabian and Oliver Gunther

The Impact of Subversive Stakeholders on Software Projects

Johann Rost and Robert L. Glass

Technical Opinion**The Ethics of IT Professionals in China**

Robert M. Davison, Maris G. Martinsons, Henry W. H. Lo, and Yuan Li



COMMUNICATIONS OF THE ACM

A monthly publication of ACM Media

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

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

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 Chesaïnais
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;
SGB Council Representatives
Norman Jouppi;
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 (514) 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

STAFF

GROUP PUBLISHER
Scott E. Delman
publisher@cacm.acm.org

Executive Editor

Diane Crawford
Managing Editor
Thomas E. Lambert
Senior Editor
Andrew Rosenblum
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 Balaquit
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;
Jeanette Wing

VIEWPOINTS

Co-chairs
Susanne E. Hambrusch;
John Leslie King;
J Strother Moore
Board Members
P. Anandan; William Aspray; Stefan
Bechtold; Judith Bishop; Soumitra Dutta;
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 ACM
Editorial Board also serves as
the Editorial Board of *queue*.

CONTRIBUTED ARTICLES

Co-chairs
Al Aho and Georg Gottlob
Board Members
Yannis Bakos; Gilles Brassard; Alan Bundy;
Peter Buneman; Ghezzi Carlo;
Andrew Chien; Anja Feldmann;
Blake Ives; James Larus; Igor Markov;
Gail C. Murphy; Shree Nayar; Lionel M. Ni;
Sriram Rajamani; Jennifer Rexford;
Marie-Christine Rousset; Avi Rubin;
Abigail Sellen; Ron Shamir; Marc Snir;
Larry Snyder; Veda Storey;
Manuela Veloso; Michael Vitale;
Wolfgang Wahlster;
Andy Chi-Chih Yao; Willy Zwaenepoel

RESEARCH HIGHLIGHTS

Co-chairs
David A. Patterson and
Stuart J. Russell
Board Members
Martin Abadi; Stuart K. Card;
Deborah Estrin; Shafi Goldwasser;
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



BPA Audit Pending

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

Annual subscription cost is included in the society member dues of \$99.00 (for students, cost is included in \$42.00 dues); the nonmember annual subscription rate is \$100.00.

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:10.1145/1538788.1538789

Open, Closed, or Clopen Access?

A frequent question I hear about *Communications*, and about ACM publishing in general, involves its access model.

I am asked: "Why don't you adopt the open-access model?" Good question! Why don't we?

Wikipedia defines open access publishing as "the publication of material in such a way that it is available to all potential users without financial or other barriers."

The Open Access movement began brewing in the 1990s, becoming fully formed with the October 2003 Berlin Declaration on Open Access to Knowledge in the Sciences and Humanities. Since then, the idea has become a hot topic in the scientific community. The *Directory of Open-Access Journals* contains over 4,000 publications. Indeed, the idea of unfettered access to scientific knowledge naturally resonates with many researchers, including me. So why doesn't ACM become an open-access publisher?

First, a point of precision. Open-access experts distinguish between "Gold OA," described earlier, and "Green OA," which allows for open access self-archiving of material (deposit by authors) that may have been published as non-open access. ACM Copyright Policy allows for self-archiving, so ACM is a Green-OA publisher. Still, why doesn't ACM become a Gold-OA publisher?

The problem with the "information wants to be free" principle is that "free," per se, is not a sound business model. The current implosion of the U.S. newspaper industry certainly testifies to that claim. Having been personally involved with an open-access publication for about five years now, I have come to re-

alize that publishing has real costs. Any publishing business model must account for these costs. Even "free" must be monetized! Google uses advertising to monetize open access, but that does not seem a viable option for scholarly publishing. Many open-access publications have adopted the "author-pays" model, requiring authors to pay thousands of dollars for each published article. The argument in favor of "author pays" is that it maximizes access to published articles, but at the same time this is simply a shifting of costs from readers to authors. Is our community ready for the author-pays model? Would this not create a new inequity between "have" and "have not" authors?

My perspective is that what really propelled the open-access movement was the continuing escalation of the price of scientific publications during the 1990s and 2000s, a period during which technology drove down the cost of scientific publishing. This price escalation has been driven by for-profit publishers. In the distant past, our field had several small- and medium-sized for-profit publishers. There was a sense of informal partnership between the scientific community and these publishers. That was then. Today, there are two large and dominant for-profit publishers in computing. These publishers are thoroughly corporatized. They are businesses with one clear mission—to maximize the return on investments to their owners and shareholders. At the same time, the scientific community, whose goal

is to maximize dissemination, continues to behave as if a partnership exists with for-profit publishers, providing them with content and editorial services essentially gratis. This is a highly anomalous arrangement, in my personal opinion. Why should for-profit corporations receive products and labor essentially for free?

As for ACM's stand on the open-access issue, I'd describe it as "clopen," somewhere between open and closed. (In topology, a clopen set is one that is both open and closed.) ACM does charge a price for its publications, but this price is very reasonable. (If you do not believe me, ask your librarian.) ACM's modest publication revenues first go to cover ACM's publication costs that go beyond print costs to include the cost of online distribution and preservation, and then to support the rest of ACM activities. To me, this is a very important point. The "profits" do not go to some corporate owners; they are used to support the activities of the association, and the association is us, the readers, authors, reviewers, and editors of ACM publications. Furthermore, ACM operates as a democratic association. If you believe that ACM should change its publishing business model, then you should lobby for this position.

The bottom line is there are two distinct issues here. The first is the issue of for-profit vs. association publishing. The current relationship between the scientific community and the for-profit publishers makes no sense to me. The second issue is the business model of association publishing, for example, "reader pays" vs. "authors pays." This is a legitimate topic of discussion, as long as we understand that it cannot be separated from the overall business model of the association. Just remember, "free" is not a sound business model.

Moshe Y. Vardi, EDITOR-IN-CHIEF

“

The quality of the editorial content, as well as the new research papers and introductions, is the reason I plan to remain an ACM member. *Communications* is a vastly better magazine as a result of these changes.

—Software vendor

”



DOI:10.1145/1538788.1538790 Scott E. Delman

***Communications'* Annual Report Card**

This issue marks the first anniversary of the completely revamped *Communications*, so I thought it would be appropriate to report on how we're doing so far. There are two

main ways to gauge the magazine's performance. The first is by asking our readers what they think of the new magazine and comparing that feedback to past results; the second is by examining actual current usage patterns, primarily online. Of course, sometimes what people tell us is different than how they really behave, so by comparing these two types of data points we can gain insight into our progress and gather enough valuable information to serve our readers even better in the future.

While this is not an exact science, I am very pleased to say that our readers response is overwhelmingly positive regarding the direction *Communications* is taking, but the proof is in the details. Over the coming months, I will share some of those details for interested readers by highlighting comments we received in recent months (some of which are peppered in these pages) and by sharing some of the up-to-date usage statistics we continue to pull off the new *Communications* Web site.

This past April, ACM conducted an extensive readership survey that was sent electronically to 5,000 of our readers around the world. It garnered a response rate of 12.16% or 608 completed surveys. Any experienced market research professional will tell you that a double-digit response rate is exceptional and is usually a strong indicator of definitive results, either positive or negative. In this case, the results are very positive. The last such survey ACM conducted several

years ago indicated that 37.9% of all respondents rated their satisfaction with the editorial focus and format of the magazine as either "satisfied" or "very satisfied." The same question posed in the most recent survey yields a result of 94.8%, a startling increase in overall satisfaction. There is, of course, an enormous amount of detail behind this general improvement in satisfaction, and for those interested, we are placing the entire survey results online at <http://cacm.acm.org/2009ReadershipSurvey.pdf>. From my own perspective, I think several key statistics are worth noting as strong indicators of a trend in ACM's membership and *Communications'* readership. They are:

“*Communications* has become a top scientific journal again, with quality standards similar to *Nature* and *Science*.”

—Researcher

► 41.3% of respondents described their current job responsibilities as Software/Applications Designer, Developer, or Engineer, followed by 23.2% as Systems Architect, Designer, or Engineer, followed by 19.9% as Academic. Indeed, we are watching an increasing slant among the magazine's readership toward practitioners and researchers in industry and the types of content that appeals to them is driving some of the changes we are making with *Communications*.

► The average reader is male (88.8%), down from 91.7% in the previous survey, 43.1 years old, down from 45.6 years old in the previous survey, has an average of 18.1 years of computing experience, down from 19.9 years in the previous survey, and has been a member of ACM for 9.8 years, down from 13.1 years in the previous survey. All of these statistics indicate that more women are entering the field (although not as quickly as many would like) and *Communications* is attracting a younger overall readership.

► The average respondent looks through 3.1 out of every 4 issues of the magazine and spends an average of 60.9 minutes reading each issue. By comparison, based on research conducted by Harvey Research, Inc. from 1996 to present, the median time spent reading business-to-business magazine titles is 38 minutes for computer titles (based on 12,500 respondents over 131 studies) and 38 minutes for non-computer titles (based on 29,700 respondents over 351 studies) with the average time spent reading over all business-to-business magazine titles being 30.7 minutes (based on 1,796 studies conducted to date over 456 different magazine titles).

► 68.6% of respondents noticed the editorial revamp of the magazine that started with the July 2008 issue and of those who noticed the change 89.9% felt it had a positive effect on the magazine.

► 58.7% of respondents read at least half of an issue's total editorial content.

► Of those who noticed the editorial revamp, 78.8% felt the changes make it more likely they would recommend the magazine to a friend and 77.3% felt the magazine is more relevant to them now than in the past.

► The most frequently read "de-

"I feel the new structure, sections, and content provide a richer experience... perhaps with a broader scope."

—Practitioner

partment" that appears in the magazine is the Editor's Letter (85.2%). The most frequently read "section" is the Research Highlights: Main Article (93.4%). And the most frequently read "column" is Viewpoints (86.4%).

► In contrast, when asked to select their three favorite departments, sections, or columns, 53.1% of respondents selected the Research Highlights: Main Article, followed by 44.7% for Contributed Articles, 35.9% for Practice articles, and 35.9% for the Research Highlights: Technical Perspectives. For the most part, what people are spending their time reading is not always their favorite material, so more investigation is certainly warranted.

► Related to the new *Communications* Web site, which launched several months ago, 46.2% of respondents were aware the Web site was being redesigned and 39.5% have visited the redesigned site. If you have not already visited the site, please do so at <http://cacm.acm.org> and login with your ACM Web Account information.

The editorial staff and editorial board for *Communications* will spend the coming months reviewing and analyzing all of the data compiled in the 2009 *Communications of the ACM* Readership Survey and in future issues will begin implementing many of the most frequently suggested changes. While the work is really just beginning, we at ACM are very pleased at the initial steps taken and very much appreciate your continued feedback and support of the Association's flagship publication.

Scott E. Delman, PUBLISHER

ACM Transactions on Accessible Computing



This quarterly publication is a quarterly journal that publishes refereed articles addressing issues of computing as it impacts the lives of people with disabilities. The journal will be of particular interest to SIGACCESS members and delegates to its affiliated conference (i.e., ASSETS), as well as other international accessibility conferences.

► ► ► ► ►
www.acm.org/taccess
www.acm.org/subscribe



Association for
Computing Machinery

Inspire with Introductory Computer Science

MARK GUZDIAL'S VIEWPOINT "Teaching Computing to Everyone" (May 2009) was interesting reading but included several implications, possibly unintentional, that should be corrected. For example, one potential benefit of contextualized computing is that it allows coursework students may find more attractive and relevant, but Guzdial seemed to imply that DrScheme and *How to Design Programs* (HtDP) cannot be used with such coursework. In our experience, this is not the case; our students are attracted and very engaged by HtDP's evolving teaching libraries. For example, students using HtDP can write interactive graphical programs from week one in a first-semester programming course without sacrificing computing fundamentals.

Libraries will soon enable them to write applications for their cellphones and embedded hardware. We look forward to experimenting with these domains in our introductory programming courses. The rich variety of contexts the HtDP community provides (and is continuously developing) excites students, and they enjoy our HtDP-based courses.

Another implication was that DrScheme and HtDP were unsuitable for non-major and female students. We found this surprising, as it is not our experience in our three very different settings. DrScheme's language levels and simple syntax seem to reduce student frustration in getting started with programming, and HtDP's design recipe approach gives them a roadmap, from problem statement and blank screen/page to a working solution. The language levels are particularly effective at reducing syntax errors by introducing new programming constructs only as the need for them arises. Both our major and non-major female students have taken quite well to this environment and approach.

Some of us are also beginning to see higher retention rates thanks to HtDP.

We were delighted to see more attention on introductory computing courses. They play a critical role in how students use, perceive, and understand computing and computer-based technology. It is important that they be well-designed, empowering students to use computing both in and outside the classroom.

Marco T. Morazan, South Orange, NJ

Marc L. Smith, Poughkeepsie, NY

Sharon Tuttle, Arcata, CA

Author's Response:

DrScheme (and its libraries) is undoubtedly one of the best programming tools for students. It inspired our Python tool, JES. To make contextualized education work, you need a language and libraries that provide the opportunity for context, a curriculum that provides examples, and lectures that support the context, as well as a course that takes advantage of these opportunities and supports. Our experience at Georgia Tech missed some of these elements. I now anticipate using DrScheme to create a great contextualized computing course.

Mark Guzdial, Atlanta, GA

More for the Practitioner, As in Web Site Design

Kudos to Steve Souders for his article "High-Performance Web Sites" (Dec. 2008). While many of the techniques he mentioned are indeed common-sense for Web site developers—reduce the number of HTTP requests and remove duplicate scripts—what impressed me most was that such a useful article made its way into *Communications* at all. In the seven years I've been a member of ACM, I've found most of its articles to be news-related or theoretical in nature. It's about time *Communications* recognized that membership includes not only researchers but also those of us keeping businesses operating by applying the theories developed in the lab and outlined in the technical literature. Please keep publishing such informa-

tive, useful articles for those of us who are practitioners.

Bryan R. Meyer, Pittsburgh, PA

To Motivate CS Students, Connect with People in Need

Two contributions (both in Apr. 2009), "Computing Education Matters" by Andrew McGetrick and "IT and the World's 'Bottom Billion'" by Richard Heeks, covered urgent problems computer scientists can help address. The former involves making computer-related education more attractive for both prospective and current students, the latter for helping the Fourth World develop itself. Students are typically of an age when altruism could be a driving force in their lives, and showing them how IT helps people in the Fourth World would add to their motivation.

To evaluate such ideas, my students and I began a project last October to provide critical information during obstetrics procedures in remote parts of sub-Saharan Africa. Obstetricians there rarely have access to current best practices, so our system gives them current information related to the APGAR scores of newborn babies. An international team of students—from Australia, China, Germany, and Switzerland—weighed the various aspects of information delivery, from usability and battery life to selective data persistence on mobile devices with limited connectivity. The project showed them how to use their knowledge and inventiveness to help others. Microsoft lent extensive support and invited them to the Imagine Cup competition. A number of NGOs also suggested ways to extend the project. We now invite *Communications* readers to participate by sharing their own ideas and imaginations.

Vladimir Stantchev, Berlin, Germany

With an Advisor Like Patterson...

Congratulations to David A. Patterson for his warm, supportive, effective model for mentoring graduate stu-

dents he explored in his "Viewpoint" "Your Students Are Your Legacy" (Mar. 2009). With appropriate changes based on the substance of study, the model is extensible well beyond CS. Patterson's legacy is indeed well deserved. I only wish he had been my advisor when I was in graduate school.

George Sadowsky, Woodstock, VT

Educating Computer Scientists About Social Science

The Viewpoint "Computing as Social Science" (Apr. 2009) by Michael Buckley was not really about social science, but about social service, which is quite a different thing. This is not a mere quibble. In 20 years of work with computer scientists, I have often had to start from the beginning, educating them about sociology—and the social sciences—as analytic disciplines.

Barry Wellman, Toronto, Canada

Cold Boot, a Surprise for Unsuspecting Users

The article "Lest We Remember: Cold-Boot Attacks on Encryption Keys" by J. Alex Halderman et al. (May 2009) took me back to my student days in the 1970s when I discovered that the Control Data Kronos operating system had a similar vulnerability. One could access other users' passwords by running the command-line tool to change passwords followed by the debug tool to "dump core" to a file. The privileged password utility could read the system password file to perform its function, but because it didn't "zero out" the RAM disk buffers before it terminated, the nonprivileged memory dump utility revealed the IDs and passwords of many other users.

Bruce Wallace, Ooltewah, TN

Equal Opportunity Support for All

You wouldn't expect a woman CS department chair and a 1960s liberal to jointly criticize an article promoting women in computing, but we were disturbed by some aspects of the cover article "Women in Computing—Take 2" (Feb. 2009).

Much of the it was devoted to a set of excellent suggestions for creating and nurturing CS careers, from initial

childhood exposure through gaining tenure at a research university. But why were these suggestions covered in an article limited to women in computing? Nearly every suggestion applies equally well to any demographic: underrepresented minorities, people with handicaps, low-income people, plain old white males. (There were a few exceptions, such as "send students to the Grace Hopper Conference" or "join CRA-W," but other career-advancing conferences and organizations can be substituted with the same overall message.) We would advise anyone considering a career in CS, or anyone in a position to nurture a CS career, to pay close attention to the good ideas in the article, while disregarding its focus on women.

For example, it suggested that introductory CS students should program in pairs. We like this idea very much for a number of reasons, none concerning gender. One might think intuitively that female students in particular prefer pair programming. However, from the statistics provided by the cited study, there is an even more positive influence on males than on females. (That is, the technique had a slightly better chance of motivating any given reluctant male to continue in CS than of motivating any given reluctant female.)

At the junior-professor level, the article suggested less teaching for the first two years, sufficient startup funding to support graduate students, help writing grant proposals, and being clear about what is expected to gain tenure. Aren't these strategies appropriate for all junior faculty? Should females be granted such departmental support while males are denied? We certainly hope not.

There's no question that women have faced obstacles over the years when choosing and building careers in CS, as well as in other fields. Still, an article providing sound general advice, while limiting it to women, is not an appropriate solution.

Jeffrey D. Ullman and Jennifer Widom, Stanford, CA

More on Browser Security

Our article "Security in the Browser" (May 2009) included a paragraph

with some unintended inaccuracies concerning the Cross Site Reference Forgery or Cross Site Request Forgery (XSRF) attack. XSRF leverages established session state in the browser. Also, if a user is authenticated into a Web site and the attacker somehow generates a URL to that site from the same browser, it may be authenticated as well. This is true for several types of authentication mechanisms, including session cookies. This type of attack does not require multiple tabs and has been around for a while, but tabs give it a new dimension, since more and more users keep multiple tabs open that are potentially authenticated to important (or high-value) sites. If a user logs into a bank and then in a separate tab goes to a page that somehow sends a malicious URL to the bank, that URL may be authenticated and able to perform actions on the user's bank account without the user's knowledge or consent. What we were attempting to show is that sometimes features have unintended security implications, an issue applicable to all major browsers.

While we regret this error, the article's original thrust is the same—that browser security issues are complex, more so every day, and the risks they pose are not to be taken lightly.

Thomas Wadlow, San Francisco, CA
Vlad Gorelik, Palo Alto, CA

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 0001-0782/09/0700 \$10.00

Coming Next Month in COMMUNICATIONS

How to glean meaning and usability from a blind user's interaction with technology.

*Boolean satisfiability:
From theoretical hardness
to practical success.*

*Revitalizing computer
education by building free
and open source software
for humanity.*

Plus the latest news on collaborative filtering, facial recognition technology, and games and education.



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:10.1145/1538788.1538792

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

Sharing Ideas, Writing Apps, and Creating a Professional Web Presence

Greg Linden reveals his new approach to reading research papers, Mark Guzdial discusses how to encourage students to write computer programs, and Tessa Lau shares her ideas about the importance of Web visibility.



From Greg Linden's "Enjoying Reading Research"

Research papers take a long time to read. They are dense, narrowly focused, often seem abstract and detached from practical issues, and occasionally require much knowledge of prior work to grasp.

Given all that work, why bother? After all, as many of my colleagues in industry say, due to the many assumptions about the quality of the data, needs of the users, performance of the algorithms, or size of the data, academic research often is unusable to them as is.

What I find most valuable about research work is that someone smart has spent a long time thinking about a particular problem. Someone has spent much effort describing a

problem, why it is important, what has been tried in the past, and what should be tried.

The authors are working to advance the state of the art, but the solution often is less valuable than the journey. For those who are trying to solve similar problems, it is the discussion of the paths taken and not taken that illuminates the road.

If you also believe this, then the way you read papers might change. Years ago, I used to turn first to the implementation and experimental results, then push the paper away if I found the evaluation lacking.

Nowadays, I turn first to the introduction, related work, conclusion, and future work. I seek to understand the problem, why it is important, what has been tried, and what still needs to be tried. I try to see why the authors chose to spend part of their

lives pursuing solutions to this task and what insights they gained. I think about how I would solve the problem myself. And only then do I turn to their solution.

Read this way, it is much easier to bask yourself in the flow of academic publications, letting the thoughts and insights wash over you. The papers become an easy joy to read, like having a conversation over coffee with the authors. It becomes what research should be, the sharing of ideas.



From Mark Guzdial's "There's an App for That," and You Could Write It"

The Apple ads for the iPhone, with the catchy phrase "There's an app for that," seem ubiquitous on television these days. They suggest that for whatever one might want to do with an iPhone, from printing a label to finding an apartment near campus, there's an application ready and waiting to help you do just that. Ready and waiting, but who made it?

One of the challenges of computing education these days is convincing students that there are new programs to write, programs that *they* want, and that they are the ones to write them. Computing is a new literacy. As Chris Crawford said, "Programming is the new writing." How do we convince students that they also want to write? It's hard to come up with a compelling argument for students that they need

real computing literacy—which isn't about using applications, and is about what is *possible* with a computer.

There are arguments that we can give students for numeracy and textual literacy. They need those things to survive in the world, because numeracy and textual literacy is pervasive in our society. But even without the pervasiveness, we can make arguments about *self-expression* and *solving one's own problems*.

We tell kids that they need to learn to write in order to write letters to their grandmothers or to write to-do lists. The fact that there have been letters written in the past is irrelevant—everyone's letter to their grandma is different. Few teachers tell kids that they should learn to write in order to become a great author (few kids will, or will even find that motivating) or that it will influence the way they think (I wonder if even all teachers believe that, though there's good cognitive science research suggesting it's true).

We do have a harder time arguing that kids should learn mathematics when they have calculators at hand. "What happens if you don't have a calculator nearby?" and "You should know how to check if your answer makes sense" are both real reasons for knowing about mathematics without a calculator, but aren't very compelling for elementary school children. The idea that mathematics might influence the way one thinks and problem-solves is again true, but not compelling for a child. Yet, the challenge to sell textual literacy or numeracy is nothing compared to the challenge of selling computational literacy.

How much harder is it to come up with a reason for coming to know computation? "There's an app for that." What should we be telling students that they can do with computation that's different or better than downloading a ready-made piece of software? What's the software equivalent of the letter to grandma, that there's a compelling reason why *your* program should be different from other programs out there? In part, the problem is a lack of imagination. As Alan Kay says, "The Computer Revolution hasn't happened yet." How do you convince kids that there's a greater

revolution possible out there and they can be part of making it happen?

So, how do these iPhone ads influence students? Do they convince students that "Every application that should be written has been written, so just buy an iPhone and don't take computer science classes"? Or do they suggest to students that "There are so many cool applications to be written. Who do you think wrote that apartment-finding application? Could be *you!*"? Do the iPhone ads suggest a universe of *possible* apps, or suggest that the applications universe is large (certainly encompassing every need *you* could possibly have), closed, fixed, and ready for download?

Reader's comment

This is a timely article for me as my school is beginning a STEM initiative and part of our goal is to convince people that computing literacy is important for every student. I am inspired to find a way for my computer science students to write an app for a G1 phone or an iPhone. That would be a very exciting assignment for them—one they would delightedly share with friends. And I can't think of a better way of "selling" an idea than letting the students do the selling to each other!

Do you have any resources that would help me learn how to write apps for mobile devices? I currently teach students Flash, Java, C programming for robots, and some Python. Finding the time to learn new things can be difficult, but this seems worthwhile.

—Debra Gouchy

Blogger's comment

Hi Debra! There's a cool class at Stanford on programming cell phones—my colleague Sarita Yardi pointed me to it. The class materials are at <http://www.stanford.edu/class/cs193p/cgi-bin/index.php>, with more of an overview at <http://studentapps.stanford.edu/>.

Some off-CACM respondents suggested to me that it's hard to make a utilitarian argument to students for programming. It might be better to think about arguing for programming as a form of expression (to build or say things that one can't easily do in any application, like with Processing, <http://www.processing.org>) or to explore ideas, like in computational science. I found both to be compelling arguments.

—Mark Guzdial



From Tessa Lau's "Visibility Matters: Why You Need a Professional Web Page"

I've been serving on a lot of selection committees in the past few years. As you get to be more senior in your field, you are tapped to participate in these committees more and more; all this volunteer work is what makes our field of endeavor possible. It's how conferences are run, papers are accepted or rejected, award winners are chosen, fellows are nominated.

If you want to succeed in this field, you need to be well known. One step you can take toward being more known is to create a Web page for yourself.

Web presence is also important at more senior levels, to select speakers for conferences, to chair a banquet, to receive an award. Chances are good you will be selected by a committee that does not know you personally. In that case, you need to have a professional Web page that gives you credibility and assures them that you are what they are looking for.

Based on my experience, here are the important details to include on your professional Web page:

- ▶ Name
- ▶ Email address
- ▶ High-level description of your research interests (e.g., HCI and AI)
- ▶ Current employer and job title
- ▶ When and where you got your Ph.D. (or when you expect to get it)
- ▶ Past and future conference responsibilities
- ▶ Conferences you have reviewed papers for
- ▶ List of representative publications
- ▶ Gender (a photo should be enough)
- ▶ Awards you have received

Many of these should be on your CV (if that isn't on the Web, it should be).

I hope I've convinced you why it's important to have a Web presence. It's particularly important for students and women in industry research labs to do this (because you tend to be less visible). Now, go update your Web site! □

Mark Guzdial is a professor at the Georgia Institute of Technology. **Greg Linden** is the founder of Geeky Ventures in Seattle, WA, and **Tessa Lau** is a research staff member at IBM Almaden Research Center.



DOI:10.1145/1538788.1538793

David Roman

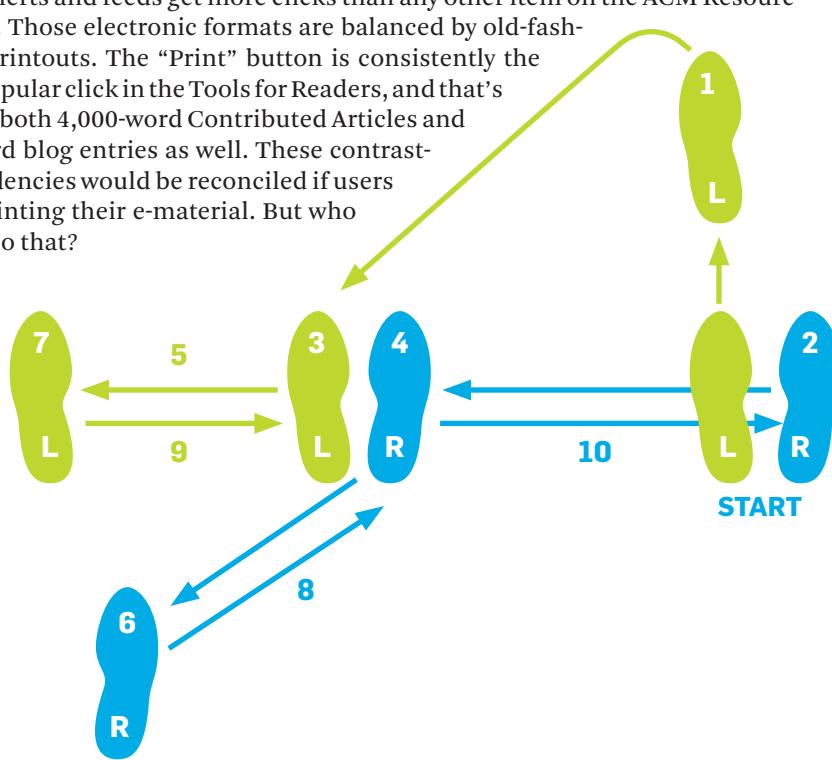
Moving Forward and Backward

My old boss was thrilled the first time he saw the image of a spinning globe online. But that was long ago, when Web users were explorers, the Internet was a place of discovery, and an animated .gif could muster boyish enthusiasm. Expectations are much higher and far more sophisticated now that users have visited hundreds of sites that demonstrate the core truth of the cliché that notes the Web is about constant change, with users like kids in a candy store pointing out cool features they'd like to see.

We can now explore some of those user expectations in the results of a Reader Profile Study conducted last April by Harvey Research Inc. for *Communications of the ACM*. The study shows that Web readers have an eye on the future and a foot in the past. Indeed, that sentiment is embodied in one reader's suggestion that ACM reintroduce self-assessment procedures and put them online. These questionnaires, designed to help a person appraise and develop his or her knowledge of a particular topic, were first launched over 30 years ago.

Other findings from the study show a split affinity for the old and the new. Half of the survey's respondents say they will use *Communications'* Web site to request RSS feeds or email alerts, fast and easy ways to get new articles. A greater number, 77.1%, will use it to access the magazine's archive of 50-plus years of articles. (For more information about this readership survey, see Scott Delman's "Publisher's Corner" on page 7.)

Recent site usage analysis reinforces the pushme-pullyu preferences of our users. Alerts and feeds get more clicks than any other item on the ACM Resources page. Those electronic formats are balanced by old-fashioned printouts. The "Print" button is consistently the most popular click in the Tools for Readers, and that's true for both 4,000-word Contributed Articles and 350-word blog entries as well. These contrasting tendencies would be reconciled if users were printing their e-material. But who would do that?



ACM Member News

GÖDEL PRIZE WINNERS

Omer Reingold, Salil Vadhan, and Avi Wigderson won the 2009 Gödel Prize for developing a new type of graph that enables the construction of large expander graphs, which play an important role in designing robust computer networks and constructing theories of error-correcting computer codes. The award, presented by ACM's Special Interest Group on Algorithms and Computing Theory and the European Association for Theoretical Computer Science, recognized their work on the zig-zag graph—a technique able to solve one of the most intriguing open problems in computational complexity theory, that of detecting a path from one node to another in very small storage for undirected graphs (in which the nodes are connected by lines with no direction).

SIGIR 09

The 32nd Annual ACM Special Interest Group on Information Retrieval (SIGIR) conference, the major international forum for the presentation of new research results and the demonstration of new systems and techniques in the field of information retrieval, will be held in Boston from July 19–23.

Networks and human behavior will be the subject of the SIGIR keynote speech by Albert-László Barabási, a professor at Northeastern University and director of its Center for Complex Network Research. "Highly interconnected networks with amazingly complex topology describe systems as diverse as the World Wide Web, our cells, social systems, or the economy," notes Barabási. "Recent studies indicate that these networks are the result of self-organizing processes governed by simple but generic laws, resulting in architectural features that make them much more similar to each other than one would have expected by chance. I will discuss the amazing order characterizing our interconnected world and its implications to network robustness and spreading processes."

Science | DOI:10.1145/1538788.1538794

Alex Wright

Contemporary Approaches to Fault Tolerance

Thanks to computer scientists like Barbara Liskov, researchers are making major progress with cost-efficient fault tolerance for Web-based systems.

AS MORE AND more data moves into the cloud, many developers find themselves grappling with the prospect of system failure at ever-widening scales.

When distributed systems first started appearing in the late 1970s and early 1980s, they typically involved a small, fixed number of servers running in a carefully managed environment. By contrast, today's Web-based distributed systems often involve thousands or hundreds of thousands of servers coming on and offline at unpredictable intervals, hosting multiple stored objects, services, and applications that often cross organizational boundaries over the Internet.

"In a cloud we have relatively few sites that are loaded with a huge number of processors," says Danny Dolev, a computer science professor at The Hebrew University of Jerusalem. "Fault tolerance needs to provide survivability and security within a cloud and across clouds."

In this deeply intertwined environment, software designers have to plan for a bewildering array of potential failure points. Building large-scale fault-tolerant systems inevitably in-

The Byzantine Generals Problem.

Figure 1
Lieutenant 2 as the Traitor

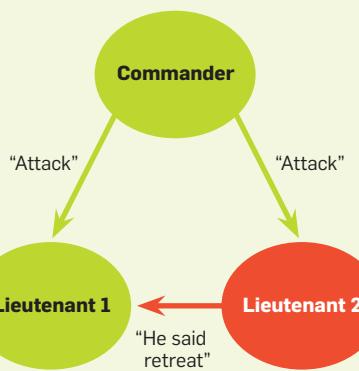
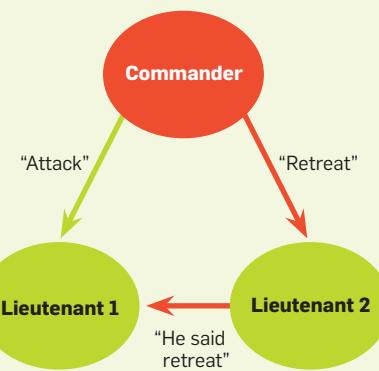


Figure 2
Commander as the Traitor



In the Byzantine Generals Problem, as defined by Leslie Lamport, Robert Shostak, and Marshall Pease in their 1982 paper, a general must communicate his order to attack or retreat to his lieutenants, but any number of participants, including the general, could be a traitor.

volves trade-offs in terms of cost, performance, and development time.

As Web systems grow, those trade-offs loom larger and larger. "Fault-tolerant systems have always been difficult to build," says University of North Carolina at Chapel Hill computer science professor Mike Reiter. "Getting a fault-tolerant system to perform as well as a non-fault-tolerant one is a challenge."

Fortunately, the research community has been making major strides in this area of late, thanks in part to the contributions of ACM A.M. Turing Award winner Barbara Liskov of Massachusetts Institute of Technology, whose breakthrough work in applying Byzantine fault tolerance (BFT) methods to the Internet has helped point the way to cost-efficient fault tolerance for Web-based systems.

While researchers have developed a number of different approaches to fault tolerance over the years, ultimately they all share a common strategy: redundancy. While hardware systems can employ redundancy at multiple levels, such as the central processing unit, memory, and firmware, fault-tolerant software design largely comes down to creating mechanisms for consistent data replication.

One of the most common approaches to software replication involves a method known as state machine replication. With state machine replication, any service provided by a computer can be described as a state machine, which accepts commands from other client machines that alter the state machine. By deploying a set of replica state machines with identical initial states, subsequent client commands can be processed by the replicas in a pre-determined fashion, so that all state machines eventually reach the same state. Thus, the failure of any one state machine can be masked by the surviving machines.

The origins of this approach to fault tolerance stretch back to the 1970s when researchers at SRI International began exploring the question of how to fly mission-critical aircraft using an assembly of computers. That work laid the foundation for contemporary approaches to fault tolerance by establishing the fundamental difference between timely systems, in which network transmission times are bounded and clocks are synchronized, and asynchronous systems, in which communication latencies have infinite-tail distribution (most messages arrive within a certain time limit but, with decreasingly low probability, messages may be delayed in transit beyond any bound).

The SRI work also helped draw important distinctions between the various types of faults experienced in a system, such as message omissions, machine crashes, or arbitrary faults due to software malfunction or other undetected data alterations. Finally, the SRI work helped to characterize resilience bounds, or how many machines are needed to tolerate certain failures.

The idea of state machine replication was given its first abstract formal-

ization by Leslie Lamport and later surveyed by Fred Schneider. Lamport's work eventually led to the Paxos protocol, a descendant of which is now in use at Google and elsewhere. Lamport used the term "Byzantine" to describe the array of possible faults that could bedevil a system. The term derives from the Byzantine Generals Problem, a logic puzzle in which a group of generals must agree on a battle plan, even though one or more of the generals may be a traitor. The challenge is to develop an effective messaging system that will outsmart the traitors and ensure execution of the battle plan. The solution, in a nutshell, involves redundancy.

While Lamport's work has proved foundational in the subsequent development of Byzantine fault tolerance, the basic ideas behind state machine replication were also implemented in other early systems. In the early 1980s, Ken Birman pursued a related line of work known as Virtual Synchrony with the ISIS system. This approach establishes rules for replication that behave indistinguishably from a non-replicated system running on a single, nonfaulty node. The ISIS approach eventually found its way into several other systems, including the CORBA fault-tolerance architecture.

At about the same time, Liskov developed viewstamped replication, a protocol designed to address benign failures, such as when a message gets lost but there's no malicious intent.

These pioneering efforts all laid the foundation for an approach to state machine replication that continues to

underlie most contemporary work on fault tolerance. However, most of these projects involved relatively small, fixed clusters of machines. "In this environment you only had to worry that the machine you stored your data on might have crashed," Liskov recalls, "but it wasn't going to tell you lies."

With the rise of the Internet in the mid-1990s, the problem of "lies"—or malicious hacks—rose to the fore. Whereas once state machines could trust each other's messages, they now had to support an additional layer of confirmation to allow for the possibility that one or more of the state machines might have been hacked.

Two groups of developers began exploring ways of applying state-machine replication techniques to cope with a growing range of Byzantine failures. Dahlia Malkhi and Mike Reiter introduced a data-centric approach known as the Byzantine quorum systems principle. In contrast to active-replication approaches like the Paxos protocol, Byzantine quorum systems focus on identifying a set of servers, rather than focusing on the messages, and choosing a set of servers so that they intersect in specific ways to ensure redundancy.

In the mid-1990s, Liskov started her breakthrough work on practical Byzantine fault tolerance (PBFT), an extension of her earlier work on viewstamped replication that adapted the Paxos replication protocol to cope with arbitrary failures. Liskov's approach demonstrated that Byzantine approaches could scale cost-effectively, sparking renewed interest in the systems research community.

While the foundational principles of consistency and replication remain essential, the rapid growth of Web systems is introducing important new challenges. Many researchers are finding that PBFT provides a useful framework for developing fault-tolerant Web systems. "I'm really excited about the recent work Barbara and her colleagues have done on making Byzantine Agreement into a practical tool—one that we can use even in large-scale settings," says Birman, a professor of computer science at Cornell University.

Inspired by Lamport and Liskov's foundational work, Hebrew University's Dolev has been working on an approach involving polynomial solutions

Practical Byzantine fault tolerance provides a useful framework for developing fault-tolerant Web systems.

to the general Byzantine agreement problem. While his early work in this area 25 years ago seemed largely theoretical, he is now finding practical applications for these approaches on the Web. "My theoretical work was ignited by Leslie [Lamport]," he says. "Barbara's work brought me to look again at the practicality of the solutions."

At Microsoft, researcher Rama Kotla has proposed a new BFT replication protocol known as Zyzzyva, that strives to improve performance by using a technique called speculation to achieve low performance overheads. Kotla is also exploring a complementary technique called high throughput BFT that exploits parallelism to improve the performance of a replicated application.

Also at Microsoft, director Chandu Thekkath has been pioneering an alternative approach to fault tolerance for Microsoft's Live Services, creating a single "configuration master" to coordinate recovery from machine failures across multiple data services. The concept of a configuration master also underlies the design of several other leading services in the live services market, such as Google's Chubby lock server.

Lorenzo Alvisi, a professor of computer science at the University of Texas at Austin, and colleagues are probing the possibilities of applying game theory techniques to fault tolerance problems, while Ittai Abraham, a professor of computer science at The Hebrew University of Jerusalem, and colleagues are incorporating security methods into distributed protocols to punish rogue participants and deter against the deviation of any collusion among them.

While these efforts are opening new research frontiers, they remain squarely rooted in the pioneering work on Byzantine fault tolerance that started more than three decades ago. Indeed, many developers are just beginning to encounter this foundational research for the first time. "Engineers are starting to discover and use these algorithms instead of writing code by the seat of their pants," says Lamport.

Many developers still wrestle with the cost and performance trade-offs of fault tolerance, however, and a number of large sites still seem willing to accept a certain degree of system failure as a

"The Web is going live," says Ken Birman. "This is going to change the picture for replication, creating a demand from average users."

cost of doing business on the Web.

"The reliability of a system increases with increasing number of tolerated failures," says Kotla, "but it also increases the cost of the system." He suggests that developers look for ways to balance costs against the need to achieve reliability in terms of mean time to failure, mean time to detect failures, and mean time to recover faulty replicas. "We need more research work in understanding and modeling faults in various settings to help system designers choose the right parameters," Kotla says.

Further complicating matters is the rise of mobile devices that are only sporadically connected to the Internet. As people entrust more and more of their personal data to these devices—like financial transactions, messaging, and other sensitive information—the challenge of keeping all that data in sync across multiple platforms will continue to escalate. And the problem of distributed fault tolerance will only grow more, well, Byzantine.

"The Web is going live," says Birman, who believes that the coming convergence of sensors, simulators, and mobile devices will drive the need for increasingly reliable data replication. "This is going to change the picture for replication, creating a demand from average users." When that happens, we may just see fault tolerance coming out of the clouds and back down to earth. □

Alex Wright is a writer and information architect who lives and works in New York City.

© 2009 ACM 0001-0782/09/0700 \$10.00

Cloud Computing

Cloning Smartphones

A pair of scientists at Intel Research Berkeley have developed CloneCloud, which creates an identical clone of an individual's smartphone that resides in a cloud-computing environment.

Created by Intel researchers Byung-Gon Chun and Petros Maniatis, CloneCloud uses a smartphone's Internet connection to communicate with the phone's online copy, which contains its data and applications, up to several gigabits in size, in the cloud. CloneCloud would make smartphones significantly faster and more powerful, enabling them to perform processor-heavy tasks in the cloud. For example, Chun and Maniatis's CloneCloud prototype, running on Google's Android mobile operating system, conducted a test application involving the facial recognition of photos. Running the application on the Android smartphone took 100 seconds; the phone's clone, operating on a desktop computer in the cloud, completed the task in one second.

According to the researchers, CloneCloud would also provide improved smartphone security, with virus scans of a device's entire file system being conducted in the cloud. Moreover, CloneCloud would improve a smartphone's battery life by having cloud-based computers handle the most processor-intensive tasks.

The CloneCloud research could help with intelligently allocating tasks to the most energy-efficient or fastest processor in a cloud-computing environment. "There will be a family of heterogeneous devices, and you would like to move the computing job to the one that makes most sense; from that standpoint, it is a great idea," said Allan Knies, associate director of Intel Research Berkeley, in an interview with *TechnologyReview*.

The CloneCloud approach could also help create a computing environment that would make it easier to share data between mobile devices and home-based computers.

Toward Native Web Execution

Several software projects are narrowing the performance gap between browser-based applications and their desktop counterparts. In the process, they're creating new ways to improve the security of Web-based computing.

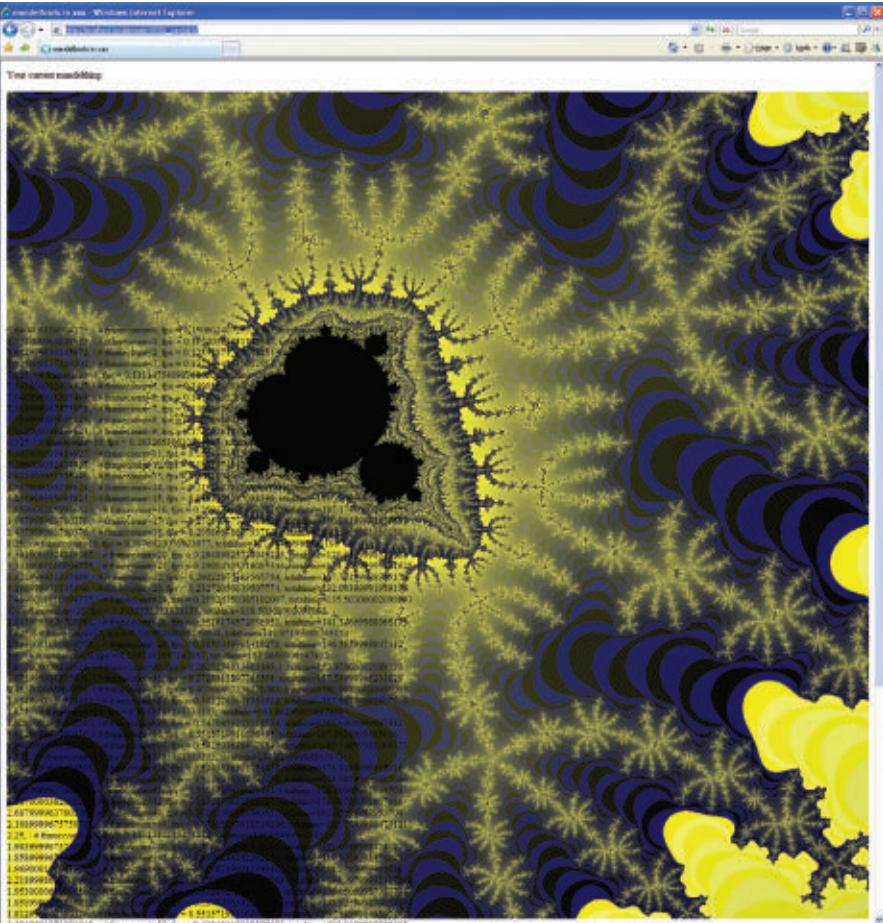
MOST INTERNET USERS do not expect the performance of browser-based applications to be the same as desktop applications, which are driven by code created from high-quality compilers and designed to run natively at the operating system (OS) level. However, several ongoing projects at Google, Microsoft, and other companies aim not only to close that performance gap, but also to eliminate some of the security weaknesses that have plagued Web browsers since the early days of the Internet.

For years, the Netscape plug-in API and Microsoft's ActiveX have provided a way to use native code modules as part of a Web application. Along with enhanced browser functionality, these extension technologies provide full access to the OS's file and networking interfaces. But by relying on trust rather than strong technical measures for safety, these extension technologies are vulnerable to social-engineering attacks in which users are tricked into permitting malicious operations.

One software project that challenges this trust model yet still offers native performance is Xax, developed at Microsoft Research. Xax separates native instruction execution from native OS access, leveraging legacy code to deliver desktop applications on the Web. The project's goal is to incorporate legacy code into browser-based applications, which then run at native performance levels and rely on a security mechanism that is more flexible than language isolation.

"Rather than use a language-based isolation mechanism, why not instead use the well-evolved and ubiquitous memory management unit?" asks researcher Jon Howell, who developed Xax at Microsoft Research.

Howell and his colleagues at Microsoft Research are currently exploring



Xax running a Mandelbrot set explorer to measure performance overhead. This CPU-bound benchmark runs as quickly inside the Xax container as when hosted in a native OS process, nearly 30 times faster than the fastest JavaScript implementations, according to Microsoft.

how a Xax interface can be used to deliver not just Web extensions, but all of a Web application's content, including the rendering functions normally provided by a browser. Realigning the client's role in this way, according to Howell, will help make browsers more secure and lead to more flexible applications that can use new rendering frameworks without forcing developers to wait for widespread client adoption.

In theory, it is possible to deliver a new codec or a variant of an HTML renderer in Flash or JavaScript. However, the new code, including all of its

libraries, would need to compile to the special language and tolerate performance penalties. "Being able to deliver native code to the client loosens the constraints," says Howell.

Different Approaches

In contrast to Xax, which relies on the memory management unit for memory isolation and a kernel system-call patch to prevent OS access, Google's Native Client takes a different approach. Using an OS-portable sandbox, Native Client relies on x86 segmentation hardware to enforce memory isolation and

on a binary validator to isolate the OS interface, preventing direct access to the OS and resources such as the file system and the network.

Despite the different implementation techniques, the idea behind Xax and Native Client is similar, according to Howell. "Let the software use the processor however it likes," he says, "and rely for isolation on a simple bit of hardware designed to do just that."

Xax and Native Client are but two of the software technologies designed to close the performance gap and strengthen the security of Web browsers. Sun's Java, Microsoft's Silverlight, and Adobe's AIR represent another approach to isolating untrusted modules from OS interfaces while narrowing the performance gap with native execution. Of course, unlike Xax and Native Client, these application frameworks tend to be used mainly as replacements for the browser-based application environment.

Another alternative approach that is gaining popularity is full virtualization. Systems such as Xen or VMware aren't commonly used to deploy Web-based applications, but that might change soon. Because virtualization systems use code-distribution formats based on native code, they avoid the performance obstacles of JavaScript and other similar languages. And to protect native OS interfaces, they wrap untrusted code in an entire instance of the OS and run that on top of simulated hardware.

"The desire is to have some kind of strong isolation barrier that an attack will not be able to penetrate," says Mendel Rosenblum, cofounder of VMware and a computer science professor at Stanford University. "Hardware-level virtual machines provide precisely that high-assurance barrier yet can run existing browsers at near-native speeds."

Rosenblum says the computer industry's focus on low-level isolation mechanisms is missing the larger point about what virtualization layers can do for performance and security, especially as the Web evolves from a document-delivery mechanism into an ecosystem of interactive applications. "The ability to run sophisticated code safely, and with high performance on the clients, will allow the new applications running in the cloud to support the richer, highly interactive interfaces users are accustomed to," he says.

Full virtualization is an alternative approach that is gaining popularity.

In the meantime, despite the proliferation of technologies that aim to sidestep the performance issues associated with running single-threaded scripts in browsers, JavaScript remains indisputably popular among developers as the only viable choice for programming browsers today. While most believe it is unlikely that JavaScript performance will catch up to the speed of native code execution, both Firefox's TraceMonkey and Google's V8, the JavaScript rendering engine in the Chrome browser, have received industrywide praise for narrowing the performance gap.

"One thing we should never lose sight of is the fact that language virtual machines are not all about straight-line speed of code and that there are many moving parts in the system that need to be balanced against each other," says Ivan Posva, a Google software engineer who developed the V8 JavaScript implementation for Chrome. Still, he says, V8 has narrowed the gap.

In terms of the next speed increase that users can expect from JavaScript rendering engines, Posva says he remains skeptical about the ability of application-specific or language-specific hardware to offer significant improvement. "Currently in V8 there are still many more optimizations that can be applied on general-purpose CPUs," he says. "I do not think that JavaScript-oriented hardware support would be a silver bullet."

In addition to the performance issue, there remains the matter of security. JavaScript running in a browser opens up the possibility for local security attacks in which a malicious application tries to elevate its privileges. "Browser designers need to be aware that the more control we give the third-party programmers via JavaScript, the more control somebody malicious could potentially have," Posva says.

"This is not a security issue on its own, but there is a lot more potential control in modern, high-performance virtual machines that can be used to exploit an independent security bug."

To mitigate these risks, V8 uses a layered approach with a sandboxed renderer. "V8 tries to minimize the attack surface by not giving total control over the generated code for a piece of JavaScript and by following common practices such as marking all data non-executable," says Posva. "V8 has to ensure that the policies set by the binding layer are followed properly."

Posva says the performance of V8 will improve regardless of whether it is embedded in a sandboxed environment. "We had to make some design decisions in V8 to allow it being embedded in the sandboxed renderer process within Google Chrome," he says. "But none of these decisions prevent a nonsandboxed use of V8, and none of these decisions had an impact on the real-world performance of V8."

That performance versatility might become increasingly important as browsers evolve, perhaps even to the point where they are no longer distinguishable from the applications they run. "In a few years," says Microsoft's Howell, "I don't think we'll mean the same thing by 'browser' that we mean today; we'll mean much less." Howell predicts that most of the functions of the traditional browser will be rendered moot, replaced by flexible code linked directly into the Web sites users visit.

Howell's prediction amounts to saying that the browser itself will become the sandbox, more or less a simple isolation framework. "Because Xax has such a narrow interface, and because we can compile the browser itself for the Xax container, you can think of Xax as a way to virtualize the browser," says Howell, who maintains that treating the host OS as something special is a short-lived phenomenon.

"As Web applications get richer, they're just as important to protect as the host OS," he says. "If Web applications are sandboxed, users can try one with no risk of exposing everything on their computer." □

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

© 2009 ACM 0001-0782/09/0700 \$10.00

Are We Losing Our Ability to Think Critically?

Computer technology has enhanced lives in countless ways, but some experts believe it might be affecting people's ability to think deeply.

SOCIETY HAS LONG cherished the ability to think beyond the ordinary. In a world where knowledge is revered and innovation equals progress, those able to bring forth greater insight and understanding are destined to make their mark and blaze a trail to greater enlightenment.

"Critical thinking as an attitude is embedded in Western culture. There is a belief that argument is the way to finding truth," observes Adrian West, research director at the Edward de Bono Foundation U.K., and a former computer science lecturer at the University of Manchester. "Developing our abilities to think more clearly, richly, fully—individually and collectively—is absolutely crucial [to solving world problems]."

To be sure, history is filled with tales of remarkable thinkers who have defined and redefined our world views: Sir Isaac Newton discovering gravity; Voltaire altering perceptions about society and religious dogma; and Albert Einstein redefining the view of the universe. But in an age of computers, video games, and the Internet, there's a growing question about how technology is changing critical thinking and whether society benefits from it.

Although there's little debate that computer technology complements—and often enhances—the human mind in the quest to store information and process an ever-growing tangle of bits and bytes, there's increasing concern that the same technology is changing the way we approach complex problems and conundrums, and making it more difficult to really *think*.

"We're exposed to [greater amounts of] poor yet charismatic thinking, the fads of intellectual fashion, opinion, and mere assertion," says West. "The wealth of communications and in-



For better or worse, exposure to technology fundamentally changes how people think.

formation can easily overwhelm our reasoning abilities." What's more, it's ironic that ever-growing piles of data and information do not equate to greater knowledge and better decision-making. What's remarkable, West says, is just "how little this has affected the quality of our thinking."

According to the National Endowment for the Arts, literary reading declined 10 percentage points from 1982 to 2002 and the rate of decline is accelerating. Many, including Patricia Greenfield, a UCLA distinguished professor of psychology and director of the Children's Digital Media Center, Los Angeles, believe that a greater focus on visual media exacts a toll. "A drop-off in reading has possibly contributed to a decline in critical thinking," she says. "There is a greater emphasis on real-time media and multitasking rather than focusing on a single thing."

Nevertheless, the verdict isn't in and a definitive answer about how technology affects critical thinking is not yet available. Instead, critical thinking lands in a mushy swamp somewhere between perception and reality; measurable and incomprehensible. It's largely a product of our own invention—and a subjective one at that. And although

technology alters the way we see, hear, and assimilate our world—the act of thinking remains decidedly human.

Rethinking Thinking

Arriving at a clear definition for critical thinking is a bit tricky. Wikipedia describes it as "purposeful and reflective judgment about what to believe or what to do in response to observations, experience, verbal or written expressions, or arguments." Overlay technology and that's where things get complex. "We can do the same critical-reasoning operations without technology as we can with it—just at different speeds and with different ease," West says.

What's more, while it's tempting to view computers, video games, and the Internet in a monolithic good or bad way, the reality is that they may be both good and bad, and different technologies, systems, and uses yield entirely different results. For example, a computer game may promote critical thinking or diminish it. Reading on the Internet may ratchet up one's ability to analyze while chasing an endless array of hyperlinks may undercut deeper thought.

Michael Bugeja, director of the Greenlee School of Journalism and Communication at Iowa State University of Science and Technology, says: "Critical thinking can be accelerated multifold by the right technology." On the other hand, "The technology distraction level is accelerating to the point where thinking deeply is difficult. We are overwhelmed by a constant barrage of devices and tasks." Worse: "We increasingly suffer from the Google syndrome. People accept what they read and believe what they see online is fact when it is not."

One person who has studied the effects of technology on people is UCLA's Greenfield. Exposure to tech-

nology fundamentally changes the way people think, says Greenfield, who recently analyzed more than 50 studies on learning and technology, including research on multitasking and the use of computers, the Internet, and video games. As reading for pleasure has declined and visual media have exploded, noticeable changes have resulted, she notes.

"Reading enhances thinking and engages the imagination in a way that visual media such as video games and television do not," Greenfield explains. "It develops imagination, induction, reflection, and critical thinking, as well as vocabulary." However, she has found that visual media actually improve some types of information processing. Unfortunately, "most visual media are real-time media that do not allow time for reflection, analysis, or imagination," she says. The upshot? Many people—particularly those who are younger—wind up not realizing their full intellectual potential.

Greenfield believes we're watching an adaptation process unfold. Today, many individuals perform better at common tasks but this doesn't make them better at thinking. The ability to multitask and use technology is highly beneficial in certain fields, including medicine, business, and flying aircraft. Consider: video game skills are a better predictor of surgeons' success in performing laparoscopic surgery than actual laparoscopic surgery experience. One study found that the best video game players made 47% fewer errors and performed 39% faster in laparoscopic tasks than the worst video game players.

"Most visual media are real-time media that do not allow time for reflection, analysis, or imagination," says Patricia Greenfield.

Tools for Learning

How society views technology has a great deal to do with how it forms perceptions about critical thinking. And nowhere is the conflict more apparent than at the intersection of video games and cognition. James Paul Gee, a professor of educational psychology at the University of Wisconsin-Madison and author of *What Video Games Have to Teach Us About Learning and Literacy*, points out that things aren't always as they appear. "There is a strong undercurrent of opinion that video and computer games aren't healthy for kids," he says. "The reality is that they are not only a major form of entertainment, they often provide a very good tool for learning."

In fact, a growing number of researchers—and an expanding body of evidence—indicate that joysticks can go a long way toward building smarter children with better reasoning skills. Games such as *Sim City*, *Civilization*, *Railroad Tycoon*, and *Age of Mythol-*

ogy extend beyond the flat earth of rote memorization and teach decision-making and analytical skills in immersive, virtual environments that resemble the real world, Gee says. Moreover, these games—and some virtual worlds—give participants freedom to explore ideas and concepts that might otherwise be inaccessible or off limits.

Kurt Squire, a University of Wisconsin-Madison associate professor in educational communications and technology, has found that as children play an educational game and learn about a particular period in history or an interesting concept, they often want to learn more. For example, one young student Squire studied sent him a list of 27 books on ancient history the boy had checked out of a library as a result of playing the game *Civilization*. What makes the games so compelling, he relates, is they create a psychological investment by "structuring problems so that they are just beyond students' current abilities."

One thing is certain. In the digital age, critical thinking is a topic that's garnering greater attention. As reading and math scores decline on standardized tests, many observers argue that it's time to take a closer look at technology and understand the subtleties of how it affects thinking and analysis. "Without critical thinking, we create trivia," Bugeja concludes. "We dismantle scientific models and replace them with trendy or wishful ones that are neither transferable nor testable." □

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

© 2009 ACM 0001-0782/09/0700 \$10.00

Milestones

Computer Science Awards

The Royal Society and the National Academy of Sciences were among the organizations that recently honored a select group of computer scientists.

THE ROYAL SOCIETY FELLOWS

Peter Buneman, a professor of database systems at the University of Edinburgh, and Dame Wendy Hall, a professor of computer science at the

University of Southampton and ACM president, were among the 44 scientists elected as Fellows of The Royal Society.

NAS MEMBERS

The National Academy of Sciences elected 72 new members and 18 foreign associates from 15 countries in recognition of their distinguished and continuing achievements in

original research. Among the new appointees are three computer scientists: Sir Timothy Berners-Lee, Massachusetts Institute of Technology; John E. Hopcroft, Cornell University; and Christos Papadimitriou, University of California, Berkeley.

SIROCCO AWARD

Nicola Santoro, a computer science professor at Carleton

University, won the Prize for Innovation in Distributed Computing from the Colloquium on Structural Information and Communication Complexity for his overall contribution on the analysis of the labeled graph properties that have been shown to have a significant impact on computability and complexity in systems of communication entities.



Liskov's Creative Joy

Barbara Liskov muses about the creative process of problem solving, finding the perfect design point, and pursuing a research path.

THE GREATEST JOY Barbara Liskov has experienced in her distinguished career has not been the results of her influential work but the creative process itself. “It’s incredibly exciting,” she says, “to be thinking about a problem and suddenly see a way to solve it that you hadn’t thought of before, and that makes a lot of other problems go away.” Creative activity is what makes research so interesting, she says, and “is not dissimilar” to what artists of all types experience during their work process. “It just happened to show up for me while thinking through solutions to problems,” she says.

Among the contributions for which Liskov received the ACM A.M. Turing Award is using data abstraction to organize software systems. This new way of thinking resulted in a paradigm shift that had immense practical consequences; it made systems much easier to build and more likely to operate correctly. It involved creating modules with an interface consisting of many operations that provided more flexibility for users than previous techniques and also allowed more details of the implementation to be hidden.

The work that led to this insight began in 1971 when Liskov was at Mitre Corporation building VENUS, a small, interactive timesharing system. She left to join Massachusetts Institute of Technology, and during the transition began reflecting on what she had accomplished with VENUS. “I stood back and thought about programming methodology and what I did in organizing the system. I saw there was this different technique being used,” she says.

This major discovery was the basis of a sequence of advances that refined and extended these ideas, Liskov says. First she saw that multi-operation modules could be naturally linked to programming languages as a way to define new data types. Working with

her research team, she built on this insight to design the programming language CLU. She decided to design a programming language because she wanted everything to be well-defined and such precision is necessary for programming languages because they are mathematical artifacts. Liskov also thought presenting the idea of data abstraction in the context of a programming language would make it easier to communicate to programmers. Additionally, Liskov firmed up the separation between how a data abstraction was implemented in a programming language and how it was described in a specification. Later, she developed the Liskov substitution principle, which explains how hierarchies of data types should be organized.

During the early 1980s, Liskov became interested in the ARPANET, the precursor to the Internet. Only a few major universities and a small group of people were using it for email and file transfers, but computer scientists dreamed of building programs that worked on a collection of ARPANET-connected machines. No one knew how to do that, so Liskov decided to

tackle the problem. While working on CLU, she consciously had limited her work to sequential programs as opposed to concurrent ones with many parts running in parallel. “We had enough problems without thinking about concurrency,” she explains, “but I had always planned as a next step to return to concurrency.” The result was the language Argus, which enables coders to write programs with components on different computers that communicated remotely through the fledgling Internet.

A stream of related work followed. Liskov delved into other aspects of distributed computing, particularly how to store files online instead of on an individual’s machines. That, in turn raised questions about crashes and losing information. Liskov worked on highly reliable storage on remote machines, which piqued her interest in replication algorithms. “To solve the problem, you must have more than one machine to store data,” says Liskov, “Then you need a protocol that enables machines to keep data in sync so you always get the most recent copy. That was the precursor to my fault tolerance work. Not yet Byzantine failures—just plain old crashes.”

Asked what it was like to develop a computer language, Liskov says, “You’re trying to create something simple and yet expressive. You’re looking for the perfect design point where the mechanisms that you put into the language are powerful enough to allow people to do the things they need to do in a fairly straightforward way, and yet syntax and semantics remain simple enough that the complexity of language isn’t overwhelming. It’s very hard work to find that design point, but it’s very satisfying. It’s a lot like mathematics because you’re looking for the elegant solution.”

Reflecting on the progress of computer science in general during her career, Liskov says people were naïve

“It’s very hard work to find that [perfect] design point, but it’s very satisfying. It’s a lot like mathematics because you’re looking for the elegant solution.”

during the early years. "I worked on a language translation project," she says. "People thought they could solve that in a few years. It was easy to not understand how difficult the problems were." Nevertheless, she acknowledges tremendous progress. In the 1970s advances were made in defining certain ways of doing things and, she says, "that's what data abstraction is all about." But the major challenge still is how to build large software systems. Such huge projects contain millions of instructions and it's hard to understand something that big, and build them to be correct and organized so that they're flexible and easy to modify, she says.

Women in computing have also made progress, although they continue to encounter unconscious gender bias, Liskov says. As associate provost for faculty equity, she educates colleagues, including members of search committees, about unconscious bias. She references studies of sexist hiring processes, including one that involved evaluations of résumés. When Swedish researchers changed men's and women's names on resumes, the resumes

"The question of how you know what's worth working on and what's not separates someone who's going to be really good at research and someone's who's not. There's no prescription."

with a woman's name were ranked lower than those with a man's. In another example, a symphony orchestra held auditions behind a curtain and, with the gender of the musicians being unknown, more women were offered

jobs. "Hopefully telling them makes them more sensitive and sophisticated," says Liskov, "so that they notice when a letter of recommendation compares a woman only to other women, for example." However, she says the issue involves not only the biased material that hiring committee members see, but also their bias in how they interpret it.

Liskov's advice for those wishing to pursue a career in research is to avoid taking a certain direction because it is likely to yield many published papers. Instead, she encourages following one's own star. "It's much better to go for the thing that's exciting," Liskov says. "But the question of how you know what's worth working on and what's not separates someone who's going to be really good at research and someone who's not. There's no prescription. It comes from your own intuition and judgment." □

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

© 2009 ACM 0001-0782/09/0700 \$10.00

ACM Transactions on Internet Technology



This quarterly publication encompasses many disciplines in computing—including computer software engineering, middleware, database management, security, knowledge discovery and data mining, networking and distributed systems, communications, and performance and scalability—all under one roof. TOIT brings a sharper focus on the results and roles of the individual disciplines and the relationship among them. Extensive multi-disciplinary coverage is placed on the new application technologies, social issues, and public policies shaping Internet development.



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

Master of Connections

Jon Kleinberg is honored for his pioneering research on the Web and social networking.

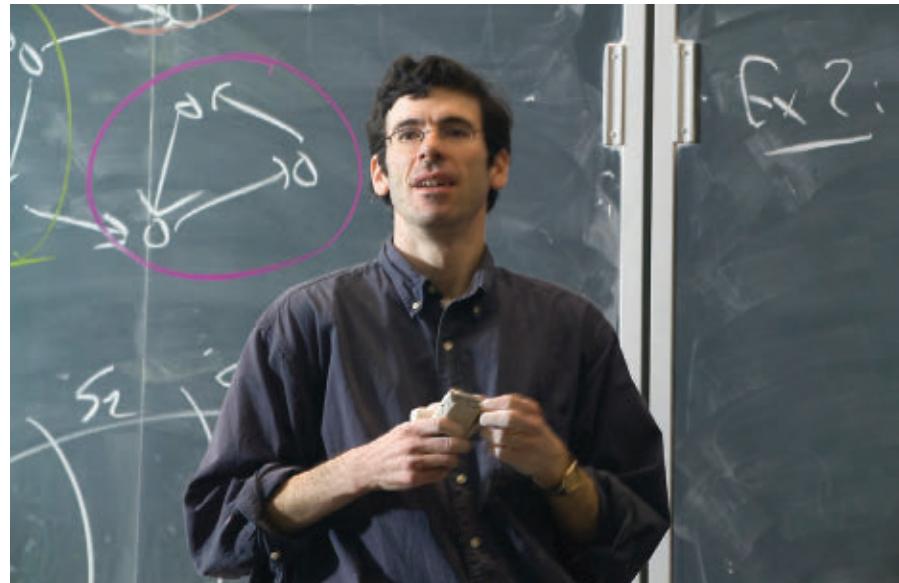
IN 1981, 10-YEAR-OLD Jon Kleinberg realized he could use his Apple II computer not just to play existing games but to invent his own. "I had a sense that you could actually create things with this device, and that presented computing in a very engaging way for me," recalls Kleinberg, now the Tisch professor of computer science at Cornell University.

That epiphany kindled in Kleinberg a passion that led him to become a rising star in computer science. The latest kudo: In April, Kleinberg won the ACM-Infosys Foundation Award in the Computing Sciences for his pioneering work in Web search techniques and large social networks. Kleinberg has previously received fellowships from the MacArthur, Packard, and Sloan foundations, and last year earned a spot on *Discover* magazine's list of "best brains under 40."

The Web link-analysis models Kleinberg created while a visiting scientist at IBM Almaden Research Center in 1996 contributed to the success of search-engine algorithms that help people navigate the volume and diversity of information on the Web, which had just exploded onto the scene a few years before. He has also used the Web's reach to explore the "six degrees of separation" phenomenon, which describes how closely connected individuals are throughout the world.

"There are problem posers, problem solvers, and problem kibitzers," says Tom Leighton, a professor of applied mathematics at Massachusetts Institute of Technology, where Kleinberg completed his graduate studies. "Jon is very good at all of the pieces. He's the kind of guy who can come up with the clever intellectual leaps and then fill in the details to prove that the ideas do work."

What makes Kleinberg's work distinctive is his ability to marry computer and social sciences. "He is driven by looking outside and then seeking



to explain it," says Susan L. Graham, computer science professor emerita at the University of California, Berkeley. "There is interesting mathematics behind what he does, but he doesn't describe it in terms of 'Here are the theorems I've proven.' He describes it in terms of 'Here's how to explain why on average there's only the distance of six hops from one person to another.'"

The ability to bridge scientific disciplines helps explain the popularity of a class Kleinberg teaches at Cornell with economist David Easley. The course,

called Networks, examines connections among social, technological, and natural worlds. "We draw from the everyday experiences of our undergrads, who are fluent in applications that enrich social connections, and ask, 'What's the science behind it?'" Kleinberg says. "That science involves computer science, economics, and the quantitative aspects of the social sciences."

Prabhakar Raghavan, head of Yahoo! Research, has seen this approach in action since 1996, when he oversaw Kleinberg's work at Almaden. One evening, they sat outside a Starbucks and watched as people ambled either into the coffee shop or into a Jamba Juice franchise next door. When Jamba Juice closed for the day and Starbucks continued to attract customers for another hour, Kleinberg quipped that Jamba Juice was losing business because it hadn't done enough data mining to understand the local market dynamics. "Jon has a very pragmatic mind," says Raghavan, "but he's always tying it back to the work he has done." □

"There are problem posers, problem solvers, and problem kibitzers," says Tom Leighton. "Jon is very good at all of the pieces."

**Previous
A.M. Turing Award
Recipients**

1966 A.J. Perlis
1967 Maurice Wilkes
1968 R.W. Hamming
1969 Marvin Minsky
1970 J.H. Wilkinson
1971 John McCarthy
1972 E.W. Dijkstra
1973 Charles Bachman
1974 Donald Knuth
1975 Allen Newell
1975 Herbert A. Simon
1976 Michael Rabin
1976 Dana Scott
1977 John Backus
1978 Robert Floyd
1979 Kenneth Iverson
1980 C.A.R. Hoare
1981 Edgar Codd
1982 Stephen Cook
1983 Ken Thompson
1983 Dennis Ritchie
1984 Niklaus Wirth
1985 Richard Karp
1986 John Hopcroft
1986 Robert Tarjan
1987 John Cocke
1988 Ivan Sutherland
1989 William Kahan
1990 Fernando Corbató
1991 Robin Milner
1992 Butler Lampson
1993 Juris Hartmanis
1993 Richard Stearns
1994 Edward Feigenbaum
1994 Raj Reddy
1995 Manuel Blum
1996 Amir Pnueli
1997 Douglas Engelbart
1998 James Gray
1999 Frederick Brooks
2000 Andrew Yao
2001 Ole-Johan Dahl
2001 Kristen Nygaard
2002 Leonard Adleman
2002 Ronald Rivest
2002 Adi Shamir
2003 Alan Kay
2004 Vinton Cerf
2004 Robert Kahn
2005 Peter Naur
2006 Frances E. Allen
2007 Edmund Clarke
2007 E. Allen Emerson
2007 Joseph Sifakis
2008 Barbara Liskov

Additional information on the past recipients of the A.M. Turing Award is available on: <http://awards.acm.org/home-page.cfm?awd=140>

ACM A.M. TURING AWARD NOMINATIONS SOLICITED

Nominations are invited for the 2009 ACM A.M. Turing Award. This, ACM's oldest and most prestigious award, is presented for contributions of a technical nature to the computing community. Although the long-term influences of the nominee's work are taken into consideration, there should be a particular outstanding technical achievement that constitutes the principal claim to the award. The award carries a prize of \$250,000 and the recipient is expected to present an address that will be published in an ACM journal. Financial support of the Turing Award is provided by the Intel Corporation and Google Inc.

Nominations should include:

- 1) A curriculum vitae, listing publications, patents, honors, other awards, etc.
- 2) A letter from the principal nominator, which describes the work of the nominee, and draws particular attention to the contribution which is seen as meriting the award.
- 3) Supporting letters from at least three endorsers. The letters should not all be from colleagues or co-workers who are closely associated with the nominee, and preferably should come from individuals at more than one organization. Successful Turing Award nominations usually include substantive letters of support from a group of prominent individuals broadly representative of the candidate's field.

**For additional information on ACM's award program
please visit: www.acm.org/awards/**

**Nominations should be sent electronically
by November 30, 2009 to:
[Alan Kay, turing@vpri.org](mailto:Alan.Kay.turing@vpri.org)**



Association for
Computing Machinery

ACM Award Winners

Among this year's distinguished honorees are Barbara Liskov of Massachusetts Institute of Technology and Jon Kleinberg of Cornell University.

EVERY YEAR ACM honors select individuals for their achievements and contributions in the areas of education, theory and practice, and service to the computing community.

ACM A.M. Turing Award

Barbara Liskov, Institute Professor, Massachusetts Institute of Technology

ACM-Infosys Foundation Award in the Computing Sciences

Jon Kleinberg, Tisch University Professor, Cornell University

Software System Award

The Gamma Parallel Database System

David J. DeWitt, Microsoft; University of Wisconsin-Madison (Emeritus)

Robert Gerber, Microsoft

Murali M. Krishna, Hewlett-Packard

Donovan A. Schneider, Yahoo!

Shahram Ghandeharizadeh, University of Southern California

Goetz Graefe, Hewlett-Packard

Michael Heytens, RGM Advisors

Hui-I Hsiao, IBM

Jeffrey F. Naughton, University of Wisconsin-Madison

Anoop Sharma, Hewlett-Packard

ACM-AAAI Allen Newell Award

Barbara J. Grosz, Higgins Professor of Natural Sciences, School of Engineering and Applied Sciences, and Dean, Radcliffe Institute for Advanced Study

Joseph Y. Halpern, Professor, Cornell University

Grace Murray Hopper Award

Dawson Engler, Associate Professor, Stanford University

Karl V. Karlstrom Outstanding Educator Award

John E. Hopcroft, IBM Professor of Engineering and Applied Mathematics, Cornell University

Paris Kanellakis Theory and Practice Award

Corinna Cortes, Head, Google Research, New York

Vladimir Vapnik, Fellow, NEC Laboratories/Columbia University

Distinguished Service Award

Telle Whitney, President and CEO, Anita Borg Institute

Outstanding Contribution to ACM Award

Wayne Graves, Director of Information Systems, ACM

Bernard Rous, Deputy Director of Publications and Electronic Publishing Program Director, ACM

ACM Fellows

Sanjeev Arora, Princeton University

Hari Balakrishnan, Massachusetts Institute of Technology

Kenneth L. Clarkson, IBM Almaden Research Center

Jason (Jingsheng) Cong, University of California at Los Angeles

Jack W. Davidson, University of Virginia

Umeshwar Dayal, Hewlett-Packard Laboratories

Xiaotie Deng, City University of Hong Kong

Jose J. Garcia-Luna-Aceves, University of California Santa Cruz; Palo Alto Research Center

Patrick Hanrahan, Stanford University

Charles H. House, Stanford University MediaX Program

Alan C. Kay, Viewpoints Research Institute

Joseph A. Konstan, University of Minnesota

Roy Levin, Microsoft Research Silicon Valley

P. Geoffrey Lowney, Intel Corporation

Jitendra Malik, University of California Berkeley

Kathryn S. McKinley, University of Texas at Austin

J. Ian Munro, University of Waterloo

Judith S. Olson, University of California at Irvine

Hamid Pirahesh, IBM Almaden Research Center

Brian Randell, Newcastle University

Michael K. Reiter, University of North Carolina at Chapel Hill

Jonathan S. Rose, University of Toronto

Mendel Rosenblum, Stanford University

Tuomas Sandholm, Carnegie Mellon University

Vivek Sarkar, Rice University

Mark S. Squillante, IBM Thomas J. Watson Research Center

Per Stenström, Chalmers University of Technology

Douglas Terry, Microsoft Research Silicon Valley

Doctoral Dissertation Award

Constantinos Daskalakis, Microsoft Research Lab

Honorable Mention:

Derek Hoiem, University of Illinois at Urbana-Champaign

Sachin Katti

ACM-W Athena Lecturer Award

Susan Eggers, Microsoft Professor of Computer Science and Engineering, University of Washington

ACM International Collegiate Programming Contest

St. Petersburg State University of IT, Mechanics and Optics

Computing Research Association Distinguished Service Award

Eugene Spafford, Professor of Computer Science, Purdue University

ACM-IEEE CS

Eckert-Mauchly Award

Joel Emer, Fellow, Intel Corporation



Congratulations ACM Senior Members

ACM honors 395 new inductees as Senior Members in recognition of their demonstrated performance which sets them apart from their peers

Greg Adamson
Kevin W. Agnew
Robert Larry Akers
Ehab Al-Shaer
Jonathan Erik Aldrich
Nega Lakew Alemayehu
Alonso Alvarez
Murali Annavaram
Phillip G. Armour
Douglas Atique
Alan F. Babich
Saurabh Bagchi
Leemon Baird
Theodore P. Baker
Dusan Baljevic
Sylvia Barnard
Alvaro Barreiro Garcia
Ira D. Baxter
James M.A. Begole
Michael P. Bekakos
Alan Berenbaum
R. Daniel Bergeron
Alessandro Berni
Pankaj Bhatt
Norbert Bieberstein
Maria Bielikova
Benjamin J. Bishop
Jean R. S. Blair
Jeffrey A. Bloom
Piero P. Bonissone
Kellogg S. Booth
Jens K.P. Borchers
Yuri Boreisha
Laszlo Bösörményi
Herve Bourlard
Guy André Boy
Wilhelm Braunschob
Duncan A. Buell
Richard Bunt
Michael A. Burns
Randal Burns
David W. Butler
Daniel K. Butler Jr.
Mario Cannataro
Robert L. Cannon
Calin Cascaval

Soumen Chatterjee
Jake Chen
Shu-Ching Chen
Zheng Chen
Ken Cheng
Gerardo Cisneros
Christina B. Class
Jack E. Cohen
Jens Coldewey
David H. Collins
Gregory Conti
Gene Cooperman
David W. Cordes
Fabio Crestani
Vladimir-Ioan Cretu
Terence Critchlow
Mark Crovella
Venu G. Dasigi
Danco Davcev
Clive B. Dawson
John D. Day
Tugrul Dayar
Paloma Diaz
Lloyd Dickman
Walter C. Dietrich, Jr.
Laura K. Dillon
Chen Ding
Mark Doernhoefer
Guozhu Dong
Jing Dong
Stefan Drees
Petros Drineas
Steven M. Drucker
Nelson F. F. Ebecken
David S. Ebert
Bo Einarsson
Samhaa R. El-Beltagy
David E. Emery
Avram Eskenazi
Opher Etzion
Claudio Feijoo
Harriet Fell
Rommel P. Feria
Bruce Filgate
Joaquim B. Filipe
Nancy Arthur Floyd

Patrick J. Flynn
Karl F. Fox
Guillermo A. Francia, III
Leo Frishberg
Antônio Augusto Fröhlich
Joachim Hans Fröhlich
Hamido Fujita
David Luigi Fuschi
Theresa Gaasterland
Samuel Henry Gamoran
Jianfeng Gao
Carlos A. Garcia
Minos Garofalakis
Marina L. Gavrilova
Edward F. Gehringer
Michael A. Gennert
Birgit Geppert
Thomas A. Gerace
Osvaldo Gervasi
Ratnanu Ghosh-Roy
Robert A. Gingell
Willard L. Graves
David Greco
Daniel C. Griffin
Georges Grinstein
William I. Grosky
Holger H. Gruen
Nuno M. Guimaraes
Neil James Gunther
Philip J. Gust
Richard A. Gustafson
Mark J. Guzdial
Martin Haenggi
Peter-Michael Hager
Sharon Hagi
Timothy J. Halloran
Paul J. Hamill, III
Dennis Hammer
Charles Hansen
Simon Harper
Lou Harrison
Robert L. Hartmann
Leslie Young Harvill
Stephen A. Harwood
Mark A. Hasegawa-Johnson
Scott Hauck

Xudong He
Milena Head
Wendi Heinzelman
Sven Helmer
Pradeep Henry
Martin Hepp
Djoerd Hiemstra
Scott A. Hissam
Aykut Hocanin
Micha Hofri
Seongsoo Hong
Ellis Horowitz
Thomas Hou
Sun-Yuan Hsieh
Xian-Sheng Hua
Charles E. Hughes
Miquel Huguet
Mikhail B. Ignatyev
John Impagliazzo
Peter Ingerman
Michael S. Irizarry
Kazuaki Ishizaki
Takayuki Ito
Sugih Jamin
Michael Jenkin
Sanjay Kumar Jha
Wang Jingfang
Jeremy R. Johnson
Magnus Jonsson
Mike Joy
Philippe Joye
Michael A. Kahn
Ejub Kajan
Surya Kant
Sumi Kaoru
Bhanu Kapoor
Alan H. Karp
Carl A. Karrfalt
Paulo Keglevich-De-Buzin
Terence Kelly
Young Hwan Kim
James C. King
Dr. Kinshuk
Peter D. Kirchner
John R. Klein
Stanislav Klimenko



Bryan T. Koch
Peter M. Kogge
Tamara G. Kolda
Fabio Kon
Hannu K. Koskela
Nectarios Koziris
M. Giridhar Krishna
Ajay D. Kshemkalyani
Amruth N. Kumar
Deepak Kumar
Yoshinori Kuno
Niels Müller Larsen
Cary Laxer
David J. Leciston
Dongman Lee
Guy Lemieux
Vitus J. Leung
Jun Li
Xiaoye S. Li
Yingjiu Li
Li Liao
Jerry Min-Chew Lim
Koon Sang Lim
Robert W. Lindeman
Tok Wang Ling
Stuart J. Lipoff
Chung-Shyan Liu
Chia-Tien Dan Lo
Julia M. Lobur
C. Douglass Locke
Antonio M. Lopez, Jr.
Karen Lopez
Manuel Lopez-Martin
David Luebke
Glenn R. Luecke
Michael R. MacFaden
Spiros Mancoridis
Sebastian Maneth
Nikolai N. Mansourov
Diana Marculescu
Timothy S. Margush
Chris Markovitch
Haralambos Marmanis
David Marston
Lutz Marten
Maurizio Martignano
J. Sperling Martin
Peter Marwedel
Michael Mascagni
Kanta Matsuura
Wolf-Ekkehard Matzke
Renee McCauley
Michael G. McKenna
Sandeep Mehta
Ronald B. Melton
Panagiotis T. Metaxas
Mira Mezini

Patrick J. Miller
Mikolay Mirenkov
Scott A. Mitchell
H. R. Mohan
Bryan R. Montgomery
Jaime H. Moreno
Nelson Morgan
John Patrick Morrison
Ali Movagher
Jörg Mühlbacher
Guenter Mueller
Debabjyoti Mukhopadhyay
John Murphy
C. R. Muthukrishnan
David W. Muttschler
Elizabeth D. Mynatt
Yunmook Nah
Zensho Nakao
V. Lakshmi Narasimhan
Leandro Navarro
Pavol Navrat
Glenn A. Nead
Michael R. Neal
Anisoara Nica
Shojoiro Nishio
Mark H. Nodine
Haruo Noma
Karl A. Nyberg
Joann J. Ordille
Thomas J. Ostrand
Barbara Boucher Owens
Jianping Pan
Gopal Pandurangan
Steven M. Paris
Joseph Pato
Trevor Pering
Luiz Felipe Perrone
Mark J. Perry
Timothy M. Pinkston
Jeremy Pitt
Vasile Podaru
George C. Polyzos
Gerard A. Pompa
Adrian Popescu
Walter D. Potter
Costin Pribeanu
Milos Prvulovic
Pradeep K. Pujari
Helen S. Raizen
Sergio Rajsbaum
Garimella Rama Murthy
Stephen Paul Ramsay
Martin Rantzer
Sandy Ressler
Vladimir V. Riabov
Kenji Rikitake
John T. Robinson

William H. Robinson
Nayan B. Ruparelia
S. Sadagopan
R. Sadananda
Maytham Hassan Safar
Deepak Sahay
Toshiaki Saisho
Bo I. Sanden
Kenya Sato
William L. Scherlis
Theo Schlossnagle
Nicu Sebe
Doree Duncan Seligmann
Everett M. Sherwood
Timothy K. Shih
Behrooz A. Shirazi
Matthew W. Silveira
Maninder Singh
Sharad K. Singhai
Anand Sivasubramaniam
Anatol Slissenko
Gregory L. Smith
James D. Smith, II
Michael K. Smith
Mario E. Sosa
William Spees
Michael Sperber
Michael Stage
J. Gregory Steffan
Fred A. Stelter
Christine Stephenson
David G. Stork
Hussein Suleman
Xian-He Sun
Jeffrey V. Sutherland
Peter F. Sweeney
Richard E. Sweet
Efstathios D. Sykas
Clarence N.W. Tan
Nicolae Tapus
Mark Tarlton
William J. Tastle
Glenn S. Tenney
Jorge Teran
Roger E. Tiptley
Satish Tiptur
Srikanta Tirthapura
Akio Tojo
Konstantin L. Tolksiy

Robert J. Torres
Thierry Turletti
Murray Tuross
John G. Tyler
George Tzanetakis
Brian William Unger
Aristides Vagelatos
Llorenç Valverde
Robert van Engelen
Sridhar Varadarajan
Giorgio Ventre
Jose M. Vidal
Ioan Orest Vlase
David W. Walker
Henry M. Walker
James Z. Wang
James (Zijun) Wang
Derek T. Warnick
Bruce W. Weide
Matt Welsh
Roberto A. Wenzel
David B. Whalley
John H. Whitehouse, Jr.
James J. Whitmore
Frank Wiegand
William B. Willaford, IV
James Reed Wilson
Robert W. Wisniewski
V.E. Wolfengagen
Bernd E. Wolfinger
Andrew Woo
Murray Woodside
Xingfu Wu
Haiping Xu
Joseph K.K. Yau
Alison L. Young
Liyun Yu
Husam Yunis
Franco Zambonelli
Alan Zeichick
Du Zhang
Liang-Jie Zhang
Lixin Zhang
Ying Zhang
Zhao Zhang
Zeljko Zilic
Detlef Zuehlke
Michael Joseph Zyda



Association for
Computing Machinery

Advancing Computing as a Science & Profession

<http://seniormembers.acm.org>



DOI:10.1145/1538788.1538800

Pamela Samuelson

Legally Speaking

The Dead Souls of the Google Book Search Settlement

Why the Google Book Search settlement agreement under consideration could result in an extensive restructuring of the book industry.

GOOGLE HAS SCANNED the texts of more than seven million books from major university research libraries for its Book Search initiative and processed the digitized copies to index their contents. Google allows users to download the entirety of these books if they are in the public domain (about one million of them are), but at this point makes available only “snippets” of relevant text when the books are still in copyright (unless the copyright owner has agreed to allow more).

In the fall of 2005, the Authors Guild, which then had about 8,000 members, and five publishers sued Google for copyright infringement. Google argued that its scanning, indexing, and snippet-providing was a fair and non-infringing use because it promoted wider public access to books and because Google would remove from its Book Search repository any digitized books whose rights holders objected to their inclusion. Many copyright professionals expected the *Authors Guild v. Google* case to be the most important fair use case of the 21st century.

This column argues that the proposed settlement of this lawsuit is a privately negotiated compulsory license primarily designed to monetize millions of orphan works. It will benefit Google and certain authors and publishers, but it is questionable whether the authors of most books in the corpus (the “dead souls” to which the column title refers) would agree that the settling authors and publishers will truly represent their interests when setting terms for access to Book Search.

Orphan Works

An estimated 70% of the books in the Book Search repository are in-copyright, but out of print. Most of them are, for all practical purposes, “orphan works,” that is, works technically still in copyright, but for which it is virtually impossible to locate the appropriate rights holders to ask for permission to digitize them.

A broad consensus exists about the desirability of making orphan works more widely available. Yet, without a safe harbor against possible infringement lawsuits, digitization projects

pose significant copyright risks. Congress is considering legislation to lessen the risk of using orphan works, but it has yet to pass.

The proposed Book Search settlement agreement solves the orphan works problem for books—at least for Google. Under this agreement, which must be approved by a federal court judge to become final, Google would get, among other things, a license to display up to 20% of the contents of in-copyright out-of-print books, to run ads alongside these displays, and to sell access to the full text of these books to institutional subscribers and individual purchasers.

The Book Rights Registry

Approval of this settlement would establish a new collecting society, the Book Rights Registry (BRR), initially funded by Google with \$34.5 million. The BRR will be responsible for allocating \$45 million in settlement funds that Google is providing to compensate copyright owners for past uses of their books.

More important is Google’s commitment to pay the BRR 63% of the revenues it makes from Book Search that are sub-

ject to sharing provisions. The revenue streams will come from ads appearing next to displays of in-copyright books in response to user queries and from individual and institutional subscriptions to some or all of the books in the corpus. Google and the BRR may also develop new business models over time that will be subject to similar sharing.

One of the main jobs of the BRR will be to distribute these revenues. The money will go, less BRR's costs, to authors and publishers who have registered their copyright claims with the BRR. Although the settlement agreement extends only to books published prior to January 5, 2009, the BRR is expected to attract authors and publishers of later-published books to participate in the revenue-sharing arrangement that Google has negotiated with the BRR.

Class Action Settlement

By now, *Communications* readers may be a bit puzzled. How can Google be getting a license to make millions of in-copyright books available through Book Search just by settling a lawsuit brought by a small fraction of authors and publishers?

How can Google be getting a license to make millions of in-copyright books available through Book Search just by settling a lawsuit?

U.S. law allows the filing of "class action" lawsuits whose lead plaintiffs claim they represent a class of persons who have suffered the same kind of harm as a result of the defendant's wrongful conduct as long as there are common issues of fact and law that make it desirable to adjudicate the claims in one lawsuit instead of many.

The Authors Guild and three of its members sued Google, claiming to represent a class of similarly situated authors whose books Google was scan-

ning and whose copyrights Google was violating. By bringing a class action lawsuit, the Authors Guild put considerable financial pressure on Google because the winner of a class action lawsuit is entitled to an award that equals all of the monies owed to the class, which may be exponentially higher than awards to individual plaintiffs.

In the absence of a settlement agreement, Google would almost certainly have vigorously fought against certification of the class in the *Authors Guild* case. After all, the guild has only a few thousand members and most of them do not write the kinds of scholarly works that are typically found in major university research libraries. Many scholarly book authors might want their books to be scanned by the Book Search project so they will be more accessible to potential readers.

The publisher lawsuit did not start out as a class action lawsuit, perhaps in part because McGraw-Hill et al. recognized how difficult it would be for them to prove they adequately represented a class of all book publishers whose books Google had scanned.

However, the agreement that Google



PHOTOGRAPH BY FLICKR USER ORANGECATS

has negotiated with the Authors Guild and the Association of American Publishers (AAP) would, if approved, be settled as a class action on behalf of *all* book authors and publishers, with the Guild and AAP claiming to represent their entire respective classes. By acceding to the certification of these classes through the settlement, Google will get a license from all authors and publishers of books covered by the agreement (which is to say nearly every in-copyright book ever published in the U.S.) so that it can commercialize them through the Book Search.

Google's New Monopoly

The proposed settlement agreement would give Google a monopoly on the largest digital library of books in the world. It and the BRR, which will also be a monopoly, will have considerable freedom to set prices and terms and conditions for Book Search's commercial services. The BRR is unlikely to complain that the price is too high, the digital rights management technology is too restrictive, or the terms are too onerous.

Google will also be the only service lawfully able to sell orphan books and monetize them through subscriptions. The BRR will get 63% of these revenues that it will pay out to registered authors and publishers, even as to books in which they hold no rights. (Some unclaimed orphan work funds may go to charities that promote literacy.) No author whose books are in the corpus can get paid by the BRR unless he or she has registered with it.

Virtually the only way that Amazon.com, Microsoft, Yahoo!, or the Open Content Alliance could get a comparably broad license as the settlement would give Google would be by starting its own project to scan books. The scanner might then be sued for copyright infringement, as Google was. It would be very costly and risky to litigate a fair use claim to final judgment given how high copyright damages may be (up to \$150,000 per infringed work). Chances are also slim that the plaintiffs in such a lawsuit would be willing or able to settle on equivalent or even similar terms.

Dead Souls

The Book Search settlement brings to mind Nikolai Gogol's story, *Dead Souls*.

The Book Search agreement under consideration is not really a settlement of a dispute over whether scanning books to index them is fair use.

Chichikov, its main character, travels around the Russian countryside to buy "dead souls" in an attempt to become a wealthy and influential man. In the early 19th century, Russian landowners had to pay annual taxes on the number of serfs—counted as "souls"—they owned as of the last census.

Chichikov offered to buy "dead souls" (serfs who had died since the last census) from the landowners. His plan was to acquire enough of these souls so that he could take out a large loan secured by his portfolio, and thereby become a wealthy man.

In Gogol's story, Chichikov's scheme falls apart. Rumors fly that the souls he owns are all dead and he flees the town in disgrace. In Google's story, however, the dead soul scheme seems likely to pay off handsomely, as Google will have the exclusive right to commercially exploit millions of orphan books.

Representativeness?

As galling as it is to realize that the BRR and its registered authors and publishers will derive income from millions of books they didn't write or publish, it is even more galling that copyright maximalists will almost certainly dominate the BRR governing board.

(The Authors Guild president, for example, complained about the "read aloud" feature of Kindle, denoting it a "swindle," and a copyright infringement. The AAP is supporting legislation to forbid the U.S. National Institutes of Health from promoting "open access" policies for articles written

under NIH grants. And of course, the Authors Guild and AAP characterized Google as a thief for scanning books from research libraries.)

If asked, authors of orphan books in major research libraries might want their books to be available under Creative Commons licenses or even be put into the public domain so that fellow researchers could have greater access to them. The BRR will have an institutional bias against encouraging this or considering what term of access most authors of books in the corpus would want.

In reviewing the settlement, the judge is supposed to consider whether the settlement is "fair" to the classes on whose behalf the lawsuits were brought. He may assume the settlement is fair because money will flow to authors and publishers. But importantly absent from the courtroom will be the orphan book authors who might have qualms about the Authors Guild and AAP as their representatives.

Conclusion

In the short run, the Google Book Search settlement will unquestionably bring about greater access to books that major research libraries collected over the years. But it is very worrisome that this agreement, which was negotiated in secret by Google and a few lawyers working for the Authors Guild and AAP (who will, incidentally, receive up to \$45.5 million in fees for their work on the settlement—more than all of the authors combined!), will create two complementary monopolies with exclusive rights over a research corpus of this magnitude. Monopolies are prone to engage in many abuses.

The Book Search agreement under consideration is not really a settlement of a dispute over whether scanning books to index them is fair use. It is a massive restructuring of the book industry's future without meaningful government oversight. The market for digitized orphan books could be competitive, but will not be if this settlement is approved in its current form without modification. C

Pamela Samuelson (pam@law.berkeley.edu) is the Richard M. Sherman Distinguished Professor of Law and Information at the University of California, Berkeley.

Copyright held by author.

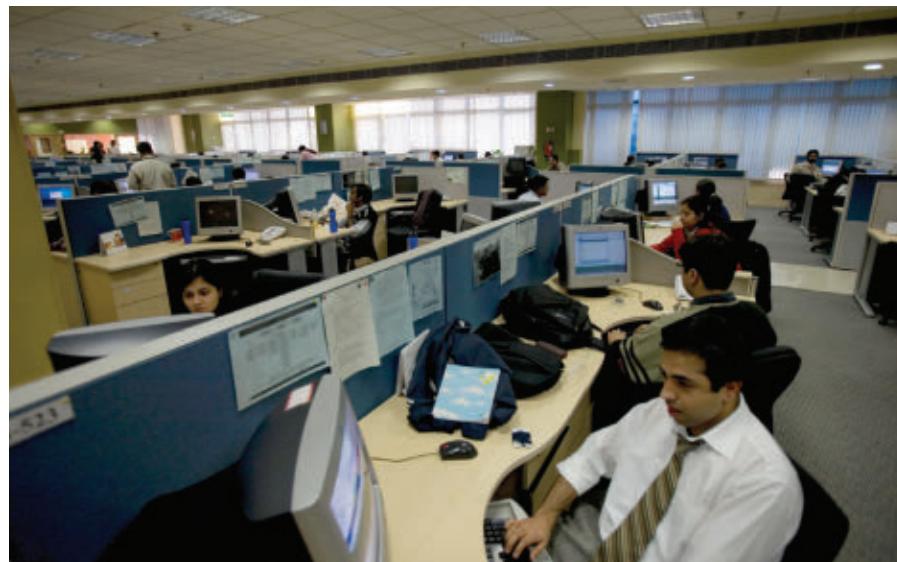
Technology Strategy and Management

Globalization of Knowledge-Intensive Professional Services

Does the trend toward standardization and modularization of professional services make outsourcing inevitable?

HIGH-END PROFESSIONAL SERVICES such as accounting and legal support are starting to move offshore in the same way that software services did a decade ago. These knowledge-intensive services are similar to software services in some respects, but different in others. It is useful to examine the reasons behind this trend and the associated implications.

Consider legal services: GE Plastics is credited with pioneering offshoring the legal support function by establishing a captive offshore base in India to draft contracts in 2001.^a Since then, the legal departments of other global corporations have followed suit. Law firms are also exploring possibilities either by establishing captive operations, as Clifford Chance had done, or by outsourcing to independent service providers. These so-called legal process outsourcing (LPO) providers are located in Indian cities like Gurgaon, Mumbai, Pune, and Hyderabad to provide legal support in patent filing, contract reviews, legal research, litigation, and compliance.^b Instead of having paralegals and contract lawyers located nearby, corporate legal departments and law firms now man-



Employees of the knowledge process outsourcing firm Evalueserve provide business and market research, data and financial analysis, and intellectual and property rights services to companies worldwide from their office in New Delhi, India.

age professionals carrying out equivalent work thousands of miles away.

Why is the offshoring of professional services—legal services in particular—occurring? The main motivator for offshoring, common across all types of services, is wage arbitrage (access to skilled labor at a fraction of the cost in the U.S. or Europe). In legal services, the hourly rate for associates in the U.S. is typically \$250–\$300, compared to approximately \$60 for U.S. paralegals and \$30 for Indian legal professionals. Offshoring is a tactic used by global cor-

porations to combat law firms' billable hour culture, which centers on the notion that costs cannot be estimated in legal work. Whereas in the past, corporate legal departments were regarded as unavoidable overheads, now they are scrutinized for more cost-effective delivery, in the same way factories have been for decades.

Behind this change in perspective is the strategy to enhance competitive advantage by unbundling corporate functions in finance, human resources, IT, procurement, marketing, and so

a Corporate Counsel, March 2003, p.78.

b Major LPO providers include CPA Global, Integreon, Evalueserve, Law-Scribe, Mindcrest, Pangea3, Quislex, and Bodhi Global.



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

forth. Both large and small enterprises can purchase professional services in these support functions off-the-shelf in global markets.² This involves the application of a global delivery model, perfected by Indian software firms for IT services, to knowledge-intensive professional services.

This all seems sensible, but will the offshore outsourcing of legal services succeed? There are major challenges to managing and capturing profit in this global value chain, including the decomposition, iteration, and disaggregation of work processes.

Work Decomposition

A prerequisite for offshore outsourcing is the breaking up of the value chain into a sequence of tasks, each with clearly defined interfaces. This poses a challenge because lawyers generally believe that decomposition in this manner may not work well. Traditionally, a client entrusted a particular lawyer to carry out an integrated service, with assistance from junior associates, paralegals, and legal secretaries. The integrated service, typically delivered in a 'job shop' craft mode, consists of at least three separable steps: knowledge and information management; consultative advice and representation; and client relationship management. For example, in litigation, document discovery (increasingly dominated by e-discovery) and legal research are part of the first step, which becomes a basis for advising and representing clients in court. Similarly, in intellectual property (IP) work, prior art search and IP portfolio analysis are part of the first step, while commercialization studies of unused patents are aspects of the second step of giving insight and advice.

In the last two decades, information and communication technology (ICT) has enabled the separation of knowledge management from advisory work. ICT has been used primarily to automate processes in data management, for example by developing document assembly software for contract drafting. Legal service, just as manufacturing, is subjected to process thinking to standardize, systematize, and package the law. Thus, the use of technology is a first step toward making legal work more repeat-

able, scalable, and offshoreable. For example, it is expected that document discovery (including e-discovery) will involve breaking a project into clearly defined components that can be worked on by separate teams in parallel. Distance should not matter in managing and coordinating these geographically dispersed teams.

In reality, there are some legal tasks, such as conveyancing, which can be easily packaged and transferred to an offshore destination. But there remain other tasks that are part of, and cannot be completely separated from, the whole. Legal research provides a good example. At one end of the spectrum, a 50-state survey on a particular issue is easily outsourceable as a standalone project. At the other end of the spectrum lies case law research that depends on knowledge of precedents and case interpretation as well as an understanding of the whole case for which research is undertaken.

Another example is document discovery in litigation support. Objective coding, involving entering the time and addressee of each email message, for example, is completely modularizable. By contrast, subjective coding, involving the identification of relevant and privileged information, cannot be done without the full knowledge of the case.

Thus, the reductionist strategy to decompose legal support work into modular parts may be difficult to implement in practice. If this sort of decomposition is easier said than done in software development, many lawyers believe, rightly or wrongly, that such things are virtually impossible for legal work. Thus, although few would doubt that some simple legal support work may be subjected to such decomposition, how much of core legal services can (and should) be decomposed in this manner remains an open question.

Work Iteration

Legal work is not performed entirely offshore, but instead the work moves back and forth between the client's home base in the U.S. or Europe and the offshore outsourcing site in India or the Philippines. This onshore/offshore mix arises out of necessity in the nature of legal work. For example, in drafting contracts, an offshore LPO

provider may create a first draft based on a template, followed by contract negotiations by the client law firm, which results in requests for adding and modifying clauses. There may be several iterations of onshore negotiations and offshore contract modifications before the final contract is produced.

Similarly in litigation support, an LPO provider may undertake document discovery for a client law firm in New York. Here, the iterative back-and-forth between the client and the LPO provider occurs due to the client's quality check on the provider's work and court-imposed deadlines for submitting specific types of documents.

Distance involved in offshore outsourcing poses a challenge to this iterative nature of work as it requires smooth handoffs and handbacks. The traditional model of doing legal work, in which an associate may walk over to instruct contract lawyers and paralegals face-to-face, is not amenable to thinking about managing the handoffs and handbacks in a systematic manner.

It was the rapid rise in legal fees that caused law firms to use more contract lawyers within the U.S. borders. But these contract lawyers, hired through staffing agencies, come and go. Some leading India-based LPO providers think that with more stable employment in India, it is easier to set up robust processes offshore than onshore, using tighter project management with milestones. Thus, ironically, U.S. law firms may hire contract lawyers located nearby on a short-term basis, while they attempt to establish longer-term stable relationships with legal professionals at a distance. It may well be that necessity is the mother of invention, and that distance is forcing LPO providers to take process control, project management, and data security more seriously. But it is not yet clear how much of legal work can be easily shifted from a traditional model to this model of process-based iteration without undermining quality.

Work Disaggregation

Offshore outsourcing will affect the way we think about professional work and the nature of professionalism itself. The shift from highly qualified to less qualified occupational skills has been

The reductionist strategy to decompose legal support work into modular parts may be difficult to implement in practice.

well under way in legal, medical, and other professional fields for reasons that have nothing to do with offshore outsourcing. However, ICT facilitates disaggregating a particular piece of work into finer standardized process steps. And the more process steps are disaggregated, the more it becomes possible to enable a non-lawyer to do legal support work. Thus, lawyers' work has become more fragmented in the same way that craftsmen were deskilled by Frederick Taylor's scientific management theory a century ago. Moreover, ICT technology further undermines the advisory function of the legal profession, as more clients rely on self-service in consuming legal services.³ Ultimately, it is the changing nature of professions onshore that enables the offshore outsourcing of professional services.

At the same time, there is an inherent pull toward keeping the profession whole, which mitigates against the de-professionalization of lawyers. In particular, some believe that even the most segmentable low-end legal work will suffer from poor quality without proper legal training. Thus, some patent attorneys may claim the knowledge of how to prosecute a patent is essential to do the most elementary aspects of patent search and drafting.

Moreover, the legal profession is self-regulated with nationally based jurisdiction. Thus, lawyers may be deemed to be less offshorable than paralegals, who in turn are less offshorable than other legal support workers precisely because the two defining characteristics of jobs that cannot be offshored apply to the legal profession.¹ So, not only does legal work require face-to-face

personal communications and/or contact with end users of the service; specific legal work must be also performed at a U.S. work location rather than overseas. Given current regulation, Indian lawyers are not permitted to practice law in the U.S. or England, while U.S. or English lawyers are not permitted to practice law in India. India-based LPO providers therefore merely supply legal support work, but never practice law in their clients' jurisdiction.

Thus, the global delivery of legal services is likely to further blur the boundary between what is done by a qualified professional and what can be done by non-qualified personnel with supervision from a qualified professional. But exactly how offshore outsourcing will affect the nature of professions is uncertain because of multiple forces at play.

Conclusion

There are several reasons why the offshore outsourcing of professional services is occurring. But there are some unknowns, especially in relation to the nature of professions that will affect the future of this phenomenon. The factors motivating offshore outsourcing are strong, and the pressures to offshore will remain. But the experience with the offshore outsourcing of software development sheds some light on just how difficult it is to deal with issues of work decomposition and iteration. The trajectory of global delivery of software work does not initially appear to translate well into that of professional services due to an additional factor of uncertainty in the nature of self-regulation of professions and the boundary of professional work. Thus, it is not advisable to draw too many conclusions about the future of the professional service offshoring from the experience thus far with software outsourcing. □

References

1. Blinder, A.S. How Many U.S. Jobs Might Be Offshorable? CEPS Working Paper No. 142, 2007.
2. Palmisano, S. The globally integrated enterprise. *Foreign Affairs* (May/June 2006).
3. Susskind, R. *The End of Lawyers?* Oxford University Press, 2008.

Mari Sako (mari.sako@sbs.ox.ac.uk) is a professor of Management Studies in Said Business School at the University of Oxford, U.K.

Copyright held by author.



DOI:10.1145/1538788.1538802

Phillip G. Armour

The Business of Software The Cliché Defense

A guide to playing the ploys frequently employed by cliché-driven management.

THERE IS A tendency exhibited by certain types of managers in certain types of organizations to manage with maxims and administer with anecdotes. Their style often consists of a warmed-over serving of the latest business self-help book garnished with an old war story and a side of the patently obvious. Such people can show a remarkable dedication to oversimplification and a common trait of this managerial style is the persistent use of the cliché.

A good cliché has several attributes:

- It covers a wide range of human behavior with just a few words.
- It sounds specific and focused but doesn't actually say much.
- It favors style over substance, pretence over production, and affect over effect.
- It has a veneer of truth that makes it plausible and difficult to argue against.
- It must suggest a solution to a problem without requiring the person

using the cliché (the cliché-er) to actually invest any energy in implementing that solution.

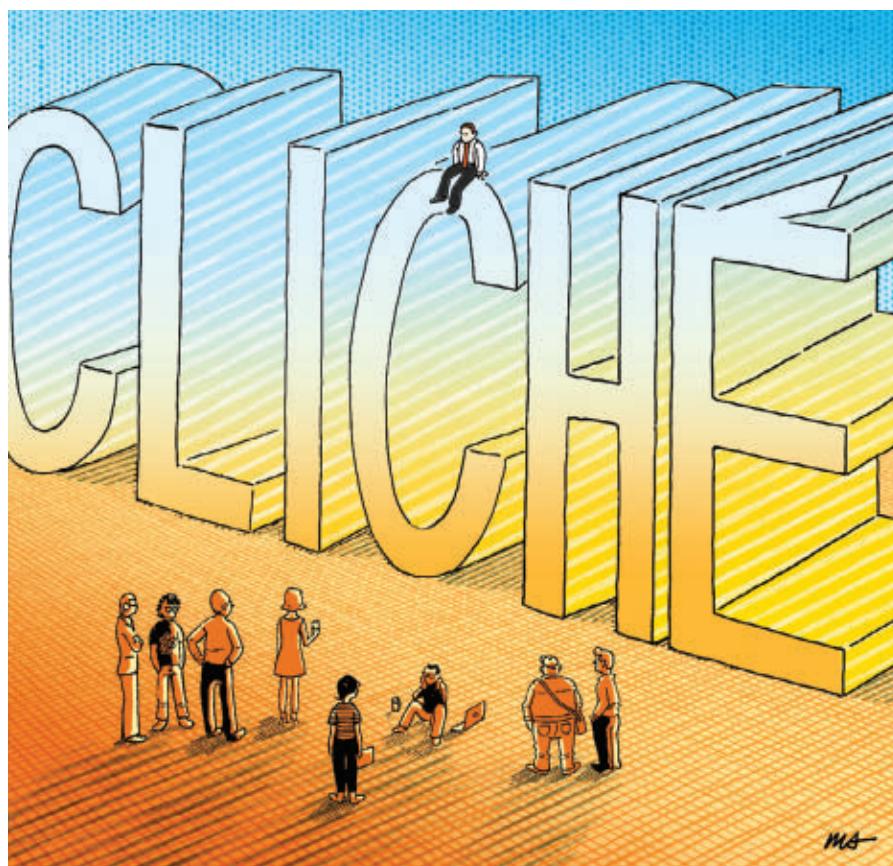
► It should leave the work of resolving the cliché to the unlucky listener (the cliché-ee). This allows any success to be claimed by the cliché-er, while locating the blame for any shortcomings in the implementation firmly on the shoulders of the unfortunate cliché-ee.

How does one defend against cliché-driven management? I have seen whole teams play the “buzzword bingo” game, gleefully tagging the hackneyed slogans of the oblivious manager. I know of senior executives in large companies who are the unwitting source of merriment for whole divisions based on their fine grasp of the obvious and their predictable production of clichés for all occasions.

The use of clichés is usually quite harmless, though it may detract from actually trying real solutions to real problems. There are legitimate defenses against certain clichés, but I must caution readers that some of these defenses use a technique called “humor.” The best humor is shared between the parties involved and reflects the comedy that exists in the situation. Use of a cliché defense as way of publicly poking fun at the person who is responsible for your continued employment has its perils.

The Cliché: “Do It Right the First Time”

Much of the business of software involves the discovery of what we are supposed to be doing. In a true discovery activity, it is only possible to not make



"mistakes" through sheer blind luck—when we just happen to hit on the optimal solution right out of the box.

Imagine walking through a dense forest for the first time. It would be almost impossible to get through the whole forest without taking a "wrong" turn. The journey must necessarily involve a certain amount of eliminating incorrect paths. Sometimes the only way to do this is to actually explore the wrong paths, because we don't know they are wrong until we try them. In fact, we could argue that the highest source of value in software development is *only* in exploring new ways to do things. If we are able to navigate through the forest without a misstep, it must be because: we have already been this way before (in which case why are we doing it again?); or we have a map (which means that *someone* has been this way before). If we can get through the forest both without error and very quickly it must be because someone has built a highway through the forest. If so, we are going toward the same destination, and building the same system, as everyone else. The real value in software is on the road less traveled, but we cannot travel this road without some exploration, and that means diverging from the path.

Inferences

You normally do it wrong, or at least it takes you many attempts to do the job properly. Clearly, you aren't smart enough to do it right without my guidance. In fact, without my leadership you aren't even smart enough to realize that you are doing it wrong at all.

Defense

If this truly is the first time, we cannot "do it right" because:

- We don't know how to do it right (because it is the first time).
- We may not even know what "right" is (because it is the first time).
- Doing it "wrong" may be the only way to find out what "right" is.
- We may actually learn more about the problem, the solution, or the business, if we do get it "wrong." An advocate for this approach was Thomas Alva Edison who was quite famous for getting it "wrong."

The Cliché: "Work Smarter, Not Harder"

This is a fine cliché, since it strongly implies that the cliché-er is actually being solicitous of the health and well-being of the cliché-ee. There are so many sneaky positional inferences in this cliché that it makes a very effective one. It has that element of truth that makes it difficult to argue against, too. Excessive work may well be a symptom of not having sufficiently thought through the problem. However, many organizations think that software is a product to be produced (rather than a knowledge medium to be populated), so the job of a software developer is to build something (rather than learn something), so the engineers should be working rather than thinking.

Inferences

It's your fault you have to work so hard, since you aren't working smarter, so you shouldn't complain about the workload. I, on the other hand, am able to see that you are not working effectively even if you cannot. Therefore, I must be smarter than you (as well as having more authority and status).

Defense

- If we were smart enough we would recognize that we aren't working smart enough. And if we were smart enough we'd be able to identify the smarter way of working and we'd also figure out how to transition from our dumb way of working to the smarter way of working.
- Therefore, if we were smart enough to figure out how to work smarter, we'd already be doing it. Clearly, we aren't smart enough to work smarter.

Much of the business of software involves the discovery of what we are supposed to be doing.

The Cliché: "Quality is the Most Important Thing"

Many organizations make this statement. Some of them even mean it. Of those companies that mean it, a few even act like they mean it. Software is a knowledge storage medium, so a defect is simply a lack of knowledge—it is something that we, as developers, did not know or did not learn and therefore didn't build into the system. So this exhortation is rather like the "work smarter" cliché.

A defining characteristic of modern software development is that the needs of the system change at close to the speed at which we can build the solution. So there may be nobody who can definitively, and in advance, determine what "perfect" is. Developers may be held accountable to a standard that no one can define.

Inferences

There is some perfect system representation that the developers *should* know but through a combination of failings (sloppiness, ignorance, laziness, ineptitude), developers have chosen to not achieve this perfection, and need to be reminded that it is important. This perfection can also be achieved without compromising any other of the "most important" attributes of the system (such as time and cost).

Defense

- Just what is "quality" and who can provide us with unequivocal guidance on it before or while we build the system (as opposed to second-guessing it after the event)?
- Are we prepared to actually do what we need to do to obtain the quality we say we need?

► If we delay delivering the system because of quality issues, then the system is 100% defective (not one part of it works, because the customer doesn't have it). Is this solution in the best interests of the customer?

Quality, along with all the other attributes of a system, is part of a balancing act. We might deliver higher quality at the cost of delayed delivery or higher cost or reduced functionality. Attaining higher quality is not a matter of just stating the goal; it usually involves discipline and hard work. And sometimes it involves difficult choices.

The Cliché: “Our Customers Are the Most Important Thing”

This cliché has that important veneer of truth. Certainly, few companies can continue in business if their customers desert them, and in the business of software delivering usable knowledge to a customer is the ultimate goal. There is ample evidence that software developers do not routinely think from the customer's perspective. But simply exhorting people to think of something important is hardly an industrial-strength business practice. Perhaps software organizations should consider building truly customer-centric development capabilities? Of course, that would be more difficult to do than just firing off a cliché.

Cliché Inferences

You need me (the cliché-er) to remind you (the cliché-ee) that we do, in fact, have customers, because left to your own devices, you software engineers would only develop what you want to: specifically the easy stuff or the “cool” stuff. Besides, software developers really think they are the most important thing.

Defense

- There are many “customers” for a system. While the paying customers are undoubtedly the “most important,” the people who test, install, support, or maintain the system are also customers.

- The value in a system is the extent to which it makes knowledge accessible and usable. The extensibility of this knowledge—how we can build on it to service future customers—is also very important. In fact, this aspect of building systems is driving the entirely appropriate focus on systems architecture and scalability we see in modern development. Few end-user customers are sophisticated enough to specifically request such features as scalability, but it is important nonetheless.

We could even argue that building the *capability* of an organization is more important than any particular customer, since it leads to the ability of the company to satisfy many more customers in the future.

The Cliché: “Our People Are the Most Important Thing”

Few clichés have more power to gen-

The real value in software is on the road less traveled, but we cannot travel this road without some exploration, and that means diverging from the path.

erate a skeptical and cynical response in its listeners than this one. There are many companies, executives, and managers who do truly believe in the people who work for them and, as far as they can, do look out for the interests of their employees. But we have probably all experienced the inflated rhetoric that sometimes passes for statements of worth and concern from executives. Its cliché-ness is not so much in the statement as in the sometimes transparent attempt at control it communicates. Most of us are quite sensitive to being manipulated like this, especially if it is done in a way that is so obvious that it also insults our intelligence.

Inferences

You (the cliché-ees) unforgivably suspect us (the cliché-ers) of wanting to manipulate you into something against your best interests. We are hurt by this lack of trust. Therefore we hope that by assuring you of our true concern for your well-being, our genuine respect for you as individuals, and our earnest desire to not have you think that we are trying to manipulate you, you will become easier to manipulate.

Defense

- If people really are the most important resource, does the company actually provide them with what they need to do the job?

- In the business of software, people are not the most important resource, they are the only resource. Software de-

velopment is a knowledge acquisition activity and the only thing that can acquire knowledge is a person. Optimizing this resource requires dealing with people honestly.

Rules of Engagement

As clear as this is, sometimes it needs to be restated. A company I once worked with adopted a set of “Rules of Engagement” intended to govern the behavior of all employees. Heading the list was “the customer is the most important thing.” One visionary company executive turned this around. He restated the imperatives as:

1. The most important thing is to build our employees’ capability.
2. The second most important thing is to build our capability to repeatedly do imperative 1.
3. The next most important thing is to deliver value to the customer.

I remember the alarm this caused, since it reversed the published order of the Rules of Engagement, but the executive was correct.

People Are the Most Important Thing

The company executive reasoned that unless you have good people working well, you simply cannot provide value to the customer. Unless you can repeatedly build and maintain your peoples’ capability, you may provide value to the customer once, but won’t be able to repeat it. And if you cannot repeat your success, you will fail your customer anyway.

This executive knew you won’t work smarter or do it right even the second or third time unless the people working in development have what they need. And you cannot act as if quality is the most important thing or the customer is the most important thing unless you first act as if your people are the most important thing. In articulating and enacting this visionary paradigm shift in their core competencies, this executive was walking the walk, going the extra mile, giving it 110%, and thinking outside of the box.

But in this case it wasn’t a cliché and that made all the difference. □

Philip G. Armour (armour@corvusintl.com) is a senior consultant at Corvus International Inc., Deer Park, IL.

Copyright held by author.

Viewpoint

Why Computer Science Doesn't Matter

Aligning computer science with high school mathematics can help turn it into an essential subject for all students.

IN MARCH 2008, the College Board (which administers the Advanced Placement (AP) exam) did the unthinkable by reducing a vibrant technology discipline, computer science, to the same level of unpopularity as a dead language, Latin. It achieved this by canceling an AP exam² in each area. Although ACM and other organizations provided data on the sustained levels of the other AP computer science exam, these statements mask the relative unpopularity of computer science compared to more traditional mathematical disciplines. Concretely, in 2007, a total of 19,392 students took one of the computer science AP exams, in contrast to 267,160 who took calculus and 96,282 who took statistics.¹

Perhaps this isn't surprising. The three Rs—reading, 'riting, 'ritmetic—symbolize what matters in U.S. primary and secondary education. Teaching these three essential skills dominates the scholastic agenda in the minds of parents, educators, and legislators. Any new material competes with these core elements; if it isn't competitive, it is marginalized.

Computer science plays such a marginal role. A large part of the problem is due to how computing is portrayed to schools, parents, the people who allocate the education budgets, and the students. The high school curriculum is mired in teaching fashionable programming languages and currently

popular programming paradigms. There is great churn in how to teach this complex content to people for whom its complexity is likely to be inappropriate. Never mind that the languages and perhaps even paradigms of today will have evaporated by the time the students graduate.

This trend is not limited to high schools; it is repeated in the introductory college curriculum. Indeed, many high schools are merely reflecting the curricular confusion at the college level. Colleges, in turn, have a problem of their own: declining enrollments in computer science.

When enrollments decline, the leaders of the computer science education community routinely look for saviors: graphics, animation, multimedia, robotics, and games have all been cast in this role. Not that integrating such topics into a course on computing is necessarily bad; but such ideas are frosting, not essentials. This search for saviors pervades thinking about introductory college curricula, and much of it percolates to thinking at the secondary school level in the form of AP and pre-AP curricula. Others, wanting to offer alternatives, act embarrassed about programming, which is our field's single most valuable skill, and seek to marginalize it (for example, see the November 2005 *Communications* column titled "Recentering Computer Science"). Meanwhile, ACM's own press releases attempt to downplay the gravity of the situation.³

What our community should really aim for is the development of a curriculum that turns our subject into the fourth R—as in 'rogramming—of our education systems. This can not only address high school curricular concerns but can also become an integral part of general education and distribution requirements in college. One way of achieving this goal is to align computing through programming with one of the three Rs and to make it indispensable. An alignment with mathematics is obvious, promising, and may even help solve some problems in mathematics education.

Mathematics and Programming

All students must enroll in mathematics for most of their school years. Many of them already struggle with it. Does hitching programming to mathematics make any sense? Consider high school algebra. Bewildering exercises about flies flitting between trains do nothing to help students understand that algebra can actually be put to work. Algebra textbooks try hard to enliven their content with high-gloss color photographs, which we can immediately recognize as symptoms of failure, not a reprieve from it. In part, school algebra appears fundamentally dull to students because it appears to be all about numbers, which play at best a small role in the media-rich, interactive lives of students. We propose the paradigm of *imaginative programming*, which weds programming to al-

gebra through the use of rich media. By embracing these media, we can engage students while synergistically meeting the needs of math teachers. Indeed, we have already seen our curricular approach, described below, help students raise their algebra scores.

How Would This Work?

Let's make this vision concrete. Algebra textbooks contain exercises that ask students to determine the next entry in a table, such as Table 1, or to create a general "variable expression" that computes any arbitrary entry of the table. In Table 1, students are expected to say that 5 comes with 25 and x comes with $x \cdot x$. We might even hope to teach the student the notation $f(x) = x \cdot x$, but why would they care? This function means nothing to them outside their homework.

We can, however, show these students that modern arithmetic and algebra do not have to be about numbers alone. They can just as well involve

images, strings, symbols, Cartesian points, and other forms of "objects." For example, Figure 1 is an arithmetic expression involving images in addition to numbers. The operator *placeImage* takes four arguments: an image (the rocket), two coordinates, and a background scene (the empty square). The value of such an expression is just another image, as shown in Figure 2. That is, algebraic expressions can both consume and compute pictorial values, enabling students to manipulate images using algebra.

Imagine asking students to determine a rising rocket's altitude after a given period of time. We could start with a table and the simplifying assumption that rockets lift off at constant speeds, as shown in Table 2. Because students understand that functions can produce images, not only numbers, we could even express this exercise as a problem involving a series of images and asking students to determine the next entry in Table 3.

By asking the student to define the function *rocket*, we are asking for a "variable expression" that computes any arbitrary entry of the table—just as we asked in the case of numbers. We would hope to get an answer like the one shown in Figure 3. A teacher may even point out here the possibility of reusing the results of one mathematical exercise in another, as shown in Figure 4. Students thus see the composition of functions and expressions, all while using mathematics as a programming language. In addition, students are motivated to learn more about mathematics and physics to improve these little programs.

With one more step, students can visualize this mathematical series of images and get the idea that constructing such mathematical series can be an aesthetically pleasing activity:

showImages(rocket, 28)

This expression demands that *rocket* be applied to 0, 1, 2, 3, 4, 5, etc., and that the result be displayed at a rate of 28 images per second. (Note how *showImages* furtively introduces the idea of functions consuming functions, because its first parameter—*rocket*—is itself a function.) Now we can tell students that making animated movies is all about using the "arithmetic of images" and its algebra.

Does It Really Work?

Readers shouldn't be surprised to find out that what we've described and illustrated here isn't just imagination or a simple software application for scripting scenes. A form of mathematics can be used as a full-fledged programming language, just like Turing Machines. In such a language, *even the design and implementation of interactive, event-driven video games doesn't take much more than algebra and geometry*. As students develop such programs they "discover" many concepts on their own simply because they want to add luster to their games—and, to formulate their improvements, they learn new mathematics and physics.

We have field-tested the beginnings of such a curriculum in the context of our TeachScheme! project for the past five years with a family of teaching languages that support images as first-class values. These languages

Table 1.

1	2	3	4	5	...	x
1	4	9	16	?	...	?

Figure 1.

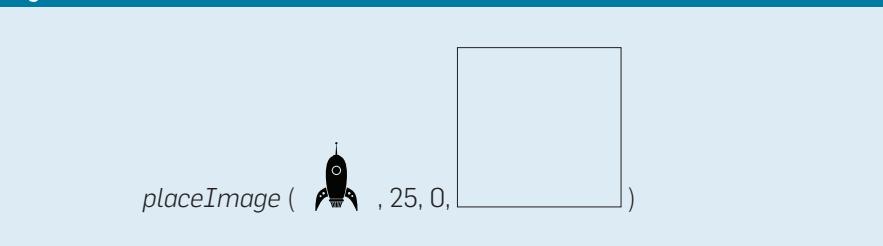


Figure 2.

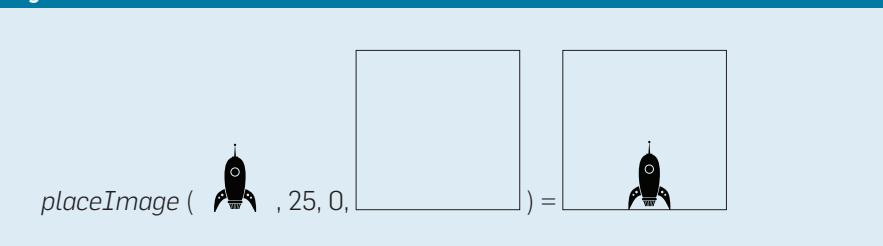


Table 2.

0	1	2	3	...	t
0	10	20	30	...	$height(t) = ?$

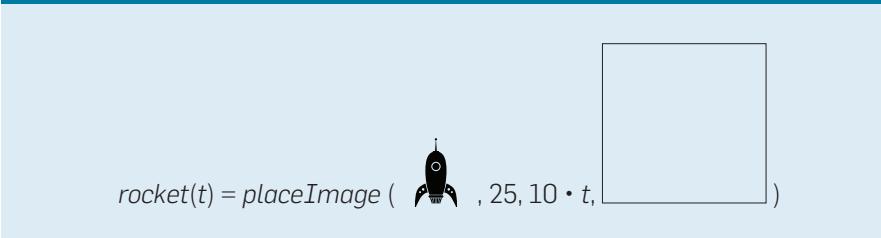
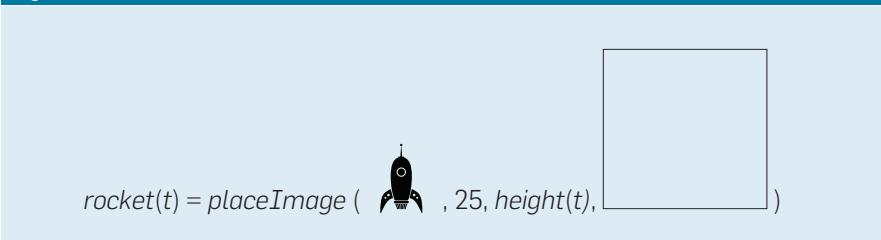
Table 3.

0	1	2	3	...	t
					$rocket(t) = ?$

are all based on Scheme, but restrict the full language to protect students from its dark corners; the languages grow in sophistication with the students' understanding. The languages are implemented in DrScheme, a pedagogic integrated development environment (IDE). The rocket simulator example described in this Viewpoint is our "Hello World" program. Students at all levels—college, high school, and middle school—react favorably to this curriculum. We also have numerous reports that students improve their performance in mathematics. In addition, formal evaluation shows the extremely positive impact this curriculum (see <http://www.teach-scheme.org/> and <http://www.bootstrapworld.org/>) has on the way educators perceive computing and programming.

At the college level, this course follows a natural progression of programming on lists, trees, documents, graphs; abstraction; programming with first-class functions and accumulators; generative recursion; stateful objects; and many more computer science concepts. We have also worked out the transition to a second course, in Java, that builds on this knowledge. Preliminary field tests validate our conjecture that the transitions are reasonably smooth and never demand a fresh start.^a

^a This material is being used in inner-city programs at several urban middle schools. It is in use at dozens of high schools. It has also been deployed at several universities of all sizes and styles in the U.S. and other countries. Representative institutions include Adelphi, Brown, Chicago, Rice, Northeastern, and Waterloo. Two German textbooks based on our material have appeared over the past two years. In India, a major corporation uses the material for its "bootcamp" for new employees. Our primary textbook has been translated into Spanish, Polish, Chinese, and (partially) German.

Figure 3.**Figure 4.**

What Makes It Work (and What Doesn't Work)

Any attempt to align programming with mathematics will fail unless the programming language is as close to school mathematics as possible. The goal of an alignment is to transfer skills from programming to mathematics and vice versa. While students quickly grasp small differences in syntax, they will mentally block if the *notion* of, say, "function" in programming significantly differs from the notion of "function" in algebra. Of course, some attributes of our approach are essential and others are accidental. We conjecture that, in addition to a language in harmony with mathematics, imaginative programming demands two more ingredients: the algebraic manipulation of images and symbolic data; and minimal overhead in the IDE for using these features.

As computer science educators, we must also demand a smooth, continuous path from imaginative programming to the engineering of large programs; otherwise beginning pro-

gramming won't create skills that transfer to our discipline. Our decade-long curricular effort has been building one such path; others may produce different transitions.

Conversely, our community must realize that minor tweaks of currently dominant approaches to programming won't suffice. Even masking the `public static void main` of Java hides little when the body of the corresponding method has little to do with the mathematical formulation of a function. The complexity of object-oriented programming bears little fruit here: it makes no sense to teach students how to engineer the structure of large systems when they are yet to write any programs with a complexity worth structuring.

Functional programming languages, such as Haskell, ML, and Scheme, suffer from different, but equally bad problems. These languages are far too complex for novices; except for DrScheme, none support images as first-class forms of data or provide pedagogical IDEs. Their type systems are

ACM Digital Library

www.acm.org/dl

The screenshot shows the ACM Digital Library homepage. It features a sidebar on the left with links to "Using the ACM Digital Library" and "FAQs". Below this is a section titled "Recently loaded issues and proceedings" listing "Journal of the ACM (JACM) Volume 52 Issue 6", "Journal on Educational Resources in Computing (ERIC) Volume 5 Issue 1", "ACM Transactions on Applied Perception (TAP) Volume 2 Issue 1", and "ACM Transactions on Software Engineering and Methodology (TOSEM) Volume 14 Issue 1". The main content area includes sections for "Advanced Search", "Browse the Digital Library" (listing Journals, Magazines, Transactions, Proceedings, Newsletters, Publications by Affiliated Organizations, and Special Interest Groups (SIGs)), "Personalized Services" (with links to My Binder and TOC Service), "Feedback" (Report a problem or Take our Satisfaction survey), and "Computing Reviews" (Access critical reviews of computing literature). There are also buttons for Join ACM, Join SIGs, and ACM Member Services.

The Ultimate Online INFORMATION TECHNOLOGY Resource!

- Over 40 ACM publications, plus conference proceedings
- 50+ years of archives
- Advanced searching capabilities
- Over 2 million pages of downloadable text

Plus over one million bibliographic citations are available in the ACM Guide to Computing Literature

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

fascinating mazes suitable for exploration by researchers and hackers, but dispatch the average student in horror after just a few interactions.

The ideal language and the IDE for imaginative programming are still to be designed. If we develop them, educational stakeholders will see how programming provides students with an interactive, engaging medium for studying and exploring mathematics. Thus, it may just turn computing into an indispensable subject for all students, right up there with the other three Rs.

Crossroads

Our community is at a crossroads when it comes to tackling our educational needs. We can continue to search for more saviors and hope that somehow, somewhere computing will receive the respect it deserves. Or we can try to help ourselves and others by turning a piece of the core school curriculum into something that students find appealing and even exciting. Our proposal is just one way of moving in this direction. We don't know whether it will succeed at large scales; and we can't know yet what else our community will discover once we start the search. What we do know is that the savior-driven ways have had their chance for many years, and they have failed.

Acknowledgments

We thank Robby Findler and Matthew Flatt for their partnership over these dozen years. Kathi Fisler helped hone our thoughts in this essay. Emmanuel Schanzer turned our ideas into the Bootstrap middle school curriculum, and his efforts have greatly influenced our thinking. C

References

1. College Board. AP: Exam Grades: Summary Reports: 2007; http://www.collegeboard.com/student/testing/ap/exgrd_sum/2007.html.
2. College Board. Important Announcement about AP Computer Science AB: Important Change for the 2009–2010 Academic Year; http://apcentral.collegeboard.com/apc/public/courses/teachers_corner/195948.html.
3. USACM. AP Computer Science is NOT Going Away; <http://usacm.acm.org/usacm/weblog/index.php?p=593>.

Matthias Felleisen (matthias@ccs.neu.edu) is Trustee Professor in the College of Computer Science at Northeastern University in Boston, MA.

Shriram Krishnamurthi (sk@cs.brown.edu) is an associate professor of computer science at Brown University in Providence, RI.

Copyright held by author.

Point/Counterpoint

CS Education in the U.S.: Heading in the Wrong Direction?

Considering the most effective methods for teaching students the fundamental principles of software engineering.

Point: Robert Dewar

LAST YEAR, EDMOND Schonberg and I published an article in *CrossTalk* (a U.S. Department of Defense software engineering journal) titled “Computer Science Education: Where Are the Software Engineers of Tomorrow?” in which we criticized the state of computer science education in U.S. universities.⁴ The article caused quite a mini-storm of discussion and was picked up by Slashdot and also by *Datamation* in an article titled “Who Killed the Software Engineer? (Hint: It Happened in College).”⁷

In our *CrossTalk* article, we expressed the general concern that the computer science curriculum was being “dumbed down” at many universities, partly in an effort to bolster declining enrollments. The enrollment decline at many universities has been dramatic, and still has not shown much sign of recovery. The twin effects of the dot-com crash and the concern of outsourcing of IT jobs seem to have convinced many parents and students that IT is not a field with a future, despite studies that project a shortage of software engineers in the near future.⁶ Perhaps the global economic meltdown will slow this cycle a bit, but I tend to agree that we will be facing a real shortage of well-trained software engineers in the future.

So obviously the question is what



do I mean by a well-trained software engineer? To me, the critical need is the knowledge required to build large complex reliable systems. It is undeniable that our society depends in a critical manner on complex software. This is not only in the familiar areas of safety-critical software like avionics systems, but also in everyday financial systems. For example, consider the report from Moody stating a bug in the Moody computer system caused an incorrect AAA rating to be assigned to \$1

billion worth of “constant proportion debt obligations.”⁵ Now I do not know exactly what this means but it is surely one of the variety of peculiar economic instruments that have been factors in the current financial crisis: the credit ratings provided by agencies such as Moody are a critical element.

I frequently give talks on safety- and security-critical software, and whenever I give such a talk, I peruse the news the week before for stories on computer security failures. Prior

to a talk last year, the high-profile stories receiving the most media attention included the break-in to vice presidential candidate Sarah Palin's email account and the successful hacking of the Large Hadron Collider Web site. Recently, one of my credit card companies reissued a card to me because a third-party database had been hacked (the credit card company would not identify the database).

I often encounter CS faculty members who take it for granted that all large computer systems are full of bugs and unreliable, and of course our experience with popular software such as Microsoft Windows reinforces this notion. The very use of the word "virus" is annoyingly misleading because it implies that really such infections are expected and impossible to eliminate, when in fact it is perfectly possible to design reliable operating systems that are immune to casual attacks. Early in the history of eBay, its auction software failed for nearly a week, and the company lost billions of dollars in capitalization. At the time I wrote to the founders of eBay that they had a company with a huge value depending on one relatively simple software application, and that there was no excuse for this application being other than entirely reliable. I commented that if their software people were telling them that such failures were inevitable, they should all be fired and replaced; I never received a reply.

So just what do we need to teach our students if they are to have the right

Undergraduate computer science curriculums simply do not regard complex software construction as a central skill to be taught.

viewpoint and skills to construct the complex reliable software systems of tomorrow, and to maintain, extend, and fix the systems in use today? In my experience, undergraduate computer science curricula simply do not regard complex software construction as a central skill to be taught. Introductory courses are dumbed down in an effort to make them fun and attractive, and have sacrificed rigor in designing and testing complex algorithms in favor of fiddling around with fun stuff such as fancy graphics. Most of these courses at this stage are using Java as a first language, and all too often Java is the only language that computer science graduates know well.

The original *CrossTalk* article was widely regarded as an anti-Java rant (one follow-up article was titled "Boffins Deride Java").⁹ It is indeed the case that the use of Java complicates basic education of programmers. It's not impossible to teach the fundamental principles using Java, but it's a difficult task. The trouble with Java is twofold. First it hides far too much, and there is far too much magic. Students using fancy visual integrated development environments working with Java end up with no idea of the fundamental structures that underlie what they are doing. Second, the gigantic libraries of Java are a seductive distraction at this level. You can indeed put together impressive fun programs just by stringing together library calls, but this is an exercise with dubious educational value. It has even been argued that it is useless to teach algorithms these days. It's as though we decided that since no one needs to know anything about how cars work, there is no point in teaching anyone the underlying engineering principles. It is vitally important that students end up knowing a variety of programming languages well and knowledge of Java libraries is not in itself sufficient.

Although the article was regarded as being anti-Java that misses the main point, which is that the curriculum lacks fundamental components that are essential in the construction of large systems. The notions of formal specification, requirements engineering, systematic testing, formal proofs of correctness, structural modeling, and so forth are typically barely pres-

It's not impossible to teach the fundamental principles using Java, but it's a difficult task.

ent in most curricula, and indeed most faculty members are not familiar with these topics, which are not seen as mainstream. For an interesting take on the importance of a practical view, see Jeff Atwood's column discussing the need to teach deployment and related practical subjects.¹

Another area of concern is that the mathematics requirements for many CS degrees have been reduced to a bare minimum. An interesting data point can be found in the construction of the iFacts system,⁸ a ground-based air-traffic control system for the U.K. that is being programmed from scratch using SPARK-Ada² and formal specification and proof of correctness techniques. It has not been easy to find programmers with the kind of mathematical skills needed to deal with formal reasoning. And yet, such formal reasoning will become an increasingly important part of software construction. As an example, consider that of the seven EAL levels of the Common Criteria for security-critical software, the top three require some level of formal reasoning to be employed.³

It is true that a lot of software development is done under conditions where reliability is not seen as critical, and the software is relatively simple and not considered as safety- or security-critical. However, if this is all we train students for then we won't have the people we need to build large complex critical systems, and furthermore this kind of simple programming is exactly the kind of job that can be successfully transferred to countries with less expensive labor costs. We are falling into a trap of training our students for outsourceable jobs.

The original article in *CrossTalk*

was based on our observations as faculty members and as software company entrepreneurs, rather than on a carefully researched study. When several people asked us for data to back up our claims, we had none to offer. Since then, however, it has been very interesting to read the flood of email we received in response to this article. In hundreds of messages, we did not get anyone saying "what are you talking about? We have no trouble hiring knowledgeable students!" On the contrary, we got hundreds of messages that said "Thank you for pointing out this problem, we find it impossible to hire competent students." One person related an experience where he had a dump from a customer for a program that had blown up and was sifting through it trying to determine what was causing the problem. A newly hired student asked him what he was doing, and he said that he was disassembling the hex into assembly language to figure out the problem. The

student, who had always considered himself superior because of his computer science degree, replied "Oh yes, assembly language, I've heard of that," and was amazed that the senior programmer (whose degree was in music) could in fact figure out the problem this way.

Another company noted that it had found it a complete waste of time to even interview graduates from U.S. universities, so they added at the end of the job description the sentence "This work will not involve Web applications or the use of Java," and that had served to almost completely eliminate U.S. applicants. Here was a case of domestic outsourcing where they were looking for people in the U.S. who had been trained in Europe and elsewhere and were better educated in the fundamentals of software engineering. These are just two examples of many similar responses, so it is clear that we have hit on a problem here that is perceived by many to be a serious one. 

References

1. Atwood, J. How should we teach computer science? *Coding Horror* (Jan. 12, 2008); <http://www.codinghorror.com/blog/archives/001035.html>.
2. Barnes, J. *High Integrity Software—The SPARK Approach to Safety and Security*. Addison-Wesley, 2003.
3. Common Criteria for Information Technology Security Evaluation, Version 3.1; September 2006; www.commoncriteriaportal.org.
4. Computer science education: Where are the software engineers of tomorrow? *CrossTalk* (Jan. 2008); http://www.stsc.hill.af.mil/CrossTalk/2008/01/0801DewarS_chonberg.html.
5. Farrell, N. Boffins deride Java. *The Inquirer* (Jan. 8, 2008); <http://www.theinquirer.net/gb/inquirer/news/2008/01/08/boffins-deride-java>.
6. Maloney, P. and Leon, M. The state of the national security space workforce (Apr. 2007); www.aero.org/publications/crosslink/spring2007/01.html.
7. McGuire, J. Who killed the software engineer? (Hint: It happened in college.) *Datamation* (Jan. 21, 2008); <http://itmanagement.earthweb.com/career/article.php/3722876>.
8. National Air Traffic Services. NATS pioneers biggest ATC advance since radar; http://www.nats.co.uk/article/218/62/nats_pioneers_biggest_atc_advance_since_radar.html.
9. Oates, J. Moody's to fix sub-prime computer error. *The Register* (July 3, 2008); http://www.theregister.co.uk/2008/07/03/moodys_computer_bug.

Robert Dewar (dewar@adacore.com) is a professor emeritus of computer science at the Courant Institute of New York University and is co-founder, president, and CEO of AdaCore.

Copyright held by author.

Counterpoint: Owen Astrachan

ROBERT DEWAR HAS graciously shouldered the task of castigating the language commonly used in introductory programming courses. Dewar, like Edsger Dijkstra⁴ and others before him, holds the language at least partially responsible for, and decries the state of, computer science curricula; he then attempts to use the programming language as a lever to move curricula in a particular direction. However, the lever of the introductory programming language is neither long enough nor strong enough to move or be responsible for our curricula. Attempts to use it as such can generate discussion, but often more heat than light. The discussion is often embroiled in fear, uncertainty, and doubt (aka FUD) rather than focused on more profound issues.

There are definite elements of FUD in the arguments offered by Dewar just as there have been by his predecessors in making similar arguments.

Whereas Dijkstra lamented "the college pretending that learning BASIC suffices or at least helps, whereas the teaching of BASIC should be rated as a criminal offense: it mutilates the mind beyond recovery" we see Dewar noting that "It's not impossible to teach the fundamental principles using Java, but it's a difficult task." Dewar and Dijkstra perhaps would like us to return to the glorious days of text editors and punch cards rather than "fancy visual IDEs." However, the slippery slope of assumption that the new generation just doesn't get it leads to the Sisyphean task of pushing the pebble of language, be it BASIC or Java, uphill against the landslide of boulders that represents the reality of computer science. This is the case regardless of whether we're in Dijkstra's world of 25 years ago, the world of 2009, or the Skynet world of tomorrow—which is probably closer than we think.

I don't mean to suggest that Dewar and Dijkstra are arguing for the same thing. Dewar would like computer science programs to produce well-trained software engineers who

can build large complex reliable systems. Dijkstra excoriated software engineering at every opportunity fixing as its charter the phrase "how to program if you cannot." Both miss part of the bigger picture in the same way that Stephen Andriole missed it in the July 2008 *Communications Point/Counterpoint* "Technology Curriculum for the Early 21st Century."¹ In his Counterpoint, Eric Roberts points out the flaw of "generalizing observations derived from one part of the field to the entire discipline." Computer science programs must embrace a far wider audience than software engineers building secure systems. Many top programs are housed in schools of Arts and Sciences rather than in Engineering, many have chosen not to be accredited by CSAB/ABET. Students may choose computer science as a stepping-stone to law, medicine, philosophy, or teaching rather than as a foundation for becoming a programmer or software engineer.

Schools like Georgia Tech are developing innovative programs to address the different needs of diverse audi-

ences: students looking to computer science as the basis for visual studies or biology rather than preparing them for a software-oriented career. There is no one-size-fits-all solution to addressing the skills and knowledge needed to succeed in these areas. Should we expect Craig Venter or Gene Myers to ask computer science programs to include more computational biology because the demand for bioinformaticians exceeds supply? Will we be surprised if Ken Perlin asks for programs to embrace games and graphics more than they do to ensure a steady supply of people interested in animation or computer-generated imagery? We are discussing the requirements and curricula of an *undergraduate* degree! Our programs can certainly build a superb foundation on which students can continue to gain knowledge and skills as they work and study in different areas, but we should no more expect students to be expert or even journeymen than we expect our premed students to be able to remove an appendix after four years of undergraduate study.

As Fred Brooks reminded us more than 20 years ago, there is no silver bullet that will solve the problems endemic to software development nor is there a panacea to cure the ills that may plague computer science curricula and programs.² Studying more mathematics will not make software bugs disappear, although both Dijkstra and Dewar seem to think so. Dewar points out the need for "formal specification and proof of correctness techniques" as foundational for software development using Ada. Dijkstra tells us "where to locate computing science on the world map of intellectual disciplines: in the direction of formal mathematics and applied logic," but pines for Algol rather than Ada. Both miss Brooks' point about the essential complexity of building software, the *essence* in the nature of software. In a wonderful treatise that has more than stood the passage of 20 years and in which he presciently anticipated the tenets of Agile software methodologies, Brooks claims that "building software will always be hard," and that this essence will not yield dramatic improvements to new languages, methodologies, or techniques.

Brooks has hopes that the essential

aspects and difficulties of software may be improved by growing software rather than building it, by buying software rather than constructing it, and by identifying and developing great designers. He differentiates between essential and accidental aspects of software where accidental is akin to incidental rather than happenstance. Changing programming languages, using MapReduce or multicore chips, and employing a visual IDE in introductory courses address these accidental or incidental parts of software development, but these don't mitigate the essential problems in developing software nor in educating our students. As Brooks notes, addressing these accidental aspects is important—high-level languages offer dramatic improvements over assembly-language programming both for software design and for introductory programming courses. Brooks' view, which I share, calls for "Hitching our research to someone else's driving problems, and solving those problems on the owners' terms, [which] leads us to richer computer science research."³ I will return to problem-driven approaches later.

It would seem from the juxtaposition of amusing anecdotes regarding flawed software systems that Dewar would like to make the academic community and the larger computer science and software communities aware that a simple change in attitude and programming language in our colleges and curricula will help make the world more secure and safe with respect to the reliable systems on which it depends. Although software runs on computers it produces outputs and ef-

fects that transcend computers. It was not a simple bug in Moody's computer system that caused constant proportion debt obligations to be incorrectly assigned the AAA rating. The model that Moody used was likely incorrectly parameterized. Even if the flaw was related to code, rather than to a model, Moody's correction of the model did not lead to a change in the AAA rating as it should have because of larger and more deeply entrenched financial and political concerns. Standard and Poor's model also assigned the AAA rating to the same constant proportion debt obligations. Both services eventually lowered their ratings, but arguably these actions were insufficient.

Blaming the current economic crisis even in part on software errors is more than a stretch. Similarly, Dewar notes that U.S. vice presidential nominee Sarah Palin's email account was compromised and that a Web site was hacked, implying these are security failures that might be fixed if only we didn't use Java in our introductory courses. Because Governor Palin used Yahoo mail for what appears to be at least semiofficial business, her password recovery mechanisms were based on publicly available information such as her birthday, and her hacked email was posted on 4chan and WikiLeaks: this is a case study in social engineering rather than one in secure systems.

Dewar's claim that Java is part of a "dumbing down" of our curricula has been echoed in other venues, notably by Joel Spolsky⁶ and Bjarne Stroustrup.⁵ However, Stroustrup notes that it isn't the language that's a problem—it is attitude. He says, and I agree that: "Education should prepare people to face new challenges; that's what makes education different from training. In computing, that means knowing your basic algorithms, data structures, system issues, etc., and the languages needed to apply that knowledge. It also means having the high-level skills to analyze a system and to experiment with alternative solutions to problems. Going beyond the simple library-user level of programming is especially important when we consider the need to build new industries, rather than just improving older ones."

Although software runs on computers it produces outputs and effects that transcend computers.

These articles, like Dewar's, associate Java with a "dumbing down" of curricula. Spolsky specifically mentions the school at which I teach as one of the new *JavaSchools*. He laments that our students are lucky in that: "The lucky kids of JavaSchools are never going to get weird segfaults trying to implement pointer-based hash tables. They're never going to go stark, raving mad trying to pack things into bits."

We didn't become a JavaSchool because we wanted to avoid segfaults, pointers, and bits. We use the same assignments and have the same attitude we did when we used C++. We switched from C++ for well-founded pedagogical reasons: Java is a better teaching language for the approach we were using than C++. Note that I'm not claiming Java is the best language for every program, but we spend much more time in our courses dealing with the Brooksian essence of programming, algorithms, and software using Java rather than with the accidental aspects symbolized by the kind of cryptic error messages that result from misusing the STL in C++. Our switch to Java was grounded neither in perceived demands from industry nor in an attempt to attract majors to our program, but in working to ensure that our beginning courses were grounded in the essence of software and algorithms.

We must work to ensure we attract motivated and capable students, not because it is incumbent on us as faculty to train the next generation of software engineers, but because it is our responsibility as educators and faculty to encourage passion and to nurture and increase the amazing opportunities that computing is bringing to our world. It is highly likely that some programming languages are better for teaching, others are better for Ajax applications, and the right flavor of Linux makes a difference. But we shortchange our students and ourselves if we live at the level of what brand of brace and bit or drill is best for a carpenter. Instead, we should look for problems that motivate the study of computing, problems that require computation in their solution.

Just as we cannot escape the essential complexity and difficulty of developing software we cannot escape the

We should look for problems that motivate the study of computing, problems that require computation in their solution.

essence of undergraduate education. We each bear the burden of our past experiences in constructing models for education. In my case this is the grounding of computer science as a liberal art, since my education began in that realm. For others, computer science is clearly an engineering discipline and to others still it is a science akin to biology or physics. We don't need to look for which of these is the correct view; they are all part of our discipline. The sooner we accept differing views as part of the whole, rather than insisting that our personally grounded view is the way to look at the world, the sooner we will make progress in crafting our curricula to meet the demands and dreams of our students. □

References

1. Andriole, S.J. and Roberts, E. Technology curriculum for the early 21st century. *Commun. ACM* 51, 7 (July 2008), 27–32.
2. Brooks, F. No silver bullet: Essence and accidents of software engineering. *IEEE Computer* 20, 4 (Apr. 1987), 10–19. Reprinted in *The Mythical Man-Month: Essays on Software Engineering*, Anniversary Edition, Addison-Wesley, 1995.
3. Brooks, F. The computer scientist as toolsmith II. *Commun. ACM* 39, 3 (Mar. 1996), 61–68.
4. Dijkstra, E. Keynote address at ACM South Central Regional Conference, Nov 16, 1984; <http://www.cs.utexas.edu/users/EWD/transcriptions/EWD08xx/EWD898.html>.
5. Maguire, J. Bjarne Stroustrup on educating software developers. *Datamation* (Dec. 9, 2008); <http://itmanagement.earthweb.com/features/article.php/3789981/>.
6. Spolsky, J. The perils of JavaSchools. Joel on Software (Dec. 29, 2005); <http://www.joelonsoftware.com/articles/ThePerilsofJavaSchools.html>.

Owen Astrachan (ola@cs.duke.edu) is Professor of the Practice of Computer Science at Duke University and the department's Director of Undergraduate Studies for Teaching and Learning.

Copyright held by author.

Calendar of Events

July 15–17
Engineering Interactive Computing Systems, Pittsburgh, PA,
Sponsored: SIGCHI,
Contact: Nicholas Graham,
Phone: 613-533-6526,
Email: graham@cs.queensu.ca

July 15–17
Symposium on Usable Privacy and Security,
Sponsored: SIGCHI,
Contact: Lorrie Faith Cranor,
Phone: 412-268-7534,
Email: lorrie@cmu.edu

July 19–23
International Symposium on Software Testing and Analysis, Chicago, IL,
Contact: Gregg E Rothermel,,
Email: grother@cs.orst.edu

July 19–23
The 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval, Boston, MA,
Contact: James Allan,,
Email: allan@cs.umass.edu

July 20–22
International Conference on Advances in Social Networks Analysis and Mining, Athens, Greece,
Contact: Nicholas Harkiolakis,
Email: nharkiolakis@hua.gr

July 23–31
Oregon Programming Languages Summer School, Eugene, OR,
Contact: Matthew T. Fluet,
Email: matthew.fluet@gmail.com

July 24–27
22nd International Conference on Industrial Engineering & Other Applications of Applied Intelligent Systems, Taiwan,
Contact: Moonis Ali,
Phone: 512-245-8050,
Email: ma04@txstate.edu

July 26–31
The 46th Annual Design Automation Conference 2009, San Francisco, CA,
Contact: Andrew B Kahng,
Phone: 858-353-0550,
Email: abk@ucsd.edu

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



Dear Colleague,

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

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

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

ACM MEMBER BENEFITS:

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

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

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

Sincerely,

Wendy Hall



President

Association for Computing Machinery



Association for
Computing Machinery



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 _____



Revisiting Gray and Putzolu's famous rule in the age of Flash.

BY GOETZ GRAEFE

The Five-Minute Rule 20 Years Later (and How Flash Memory Changes the Rules)

IN 1987, JIM Gray and Gianfranco Putzolu published their now-famous five-minute rule¹⁵ for trading off memory and I/O capacity. Their calculation compares the cost of holding a record (or page) permanently in memory with the cost of performing disk I/O each time the record (or page) is accessed, using appropriate fractional prices of RAM chips and disk drives. The name of their rule refers to the break-even interval between accesses. If a record (or page) is accessed more often, it should be kept in memory; otherwise, it should remain on disk and be read when needed.

Based on then-current prices and performance characteristics of Tandem equipment, Gray and Putzolu found the price of RAM to hold a 1KB record

was about equal to the (fractional) price of a disk drive required to access such a record every 400 seconds, which they rounded to five minutes. The break-even interval is about inversely proportional to the record size. Gray and Putzolu reported one hour for 100-byte records and two minutes for 4KB pages.

The five-minute rule was reviewed and renewed 10 years later.¹⁴ Lots of prices and performance parameters had changed (for example, the price of RAM had tumbled from \$5,000 to \$15 per megabyte). Nonetheless, the break-even interval for 4KB pages was still around five minutes. The first goal of this article is to review the five-minute rule after another 10 years.

Of course, both previous articles acknowledged that prices and performance vary among technologies and devices at any point in time (RAM for mainframes versus minicomputers, SCSI versus IDE disks, and so on). Interested readers are invited to reevaluate the appropriate formulas for their environments and equipment. The values used here (in Table 1) are meant to be typical for 2007 technologies rather than universally accurate.

In addition to quantitative changes in prices and performance, qualitative changes already under way will affect the software and hardware architectures of servers and, in particular, database systems. Database software will change radically with the advent of new technologies: virtualization with hardware and software support, as well as higher utilization goals for physical machines; many-core processors and transactional memory supported both in programming environments and hardware;²⁰ deployment in containers housing thousands of processors and many terabytes of data;¹⁷ and flash memory that fills the gap between traditional RAM and traditional rotating disks.

Flash memory falls between traditional RAM and persistent mass storage based on rotating disks in terms of acquisition cost, access

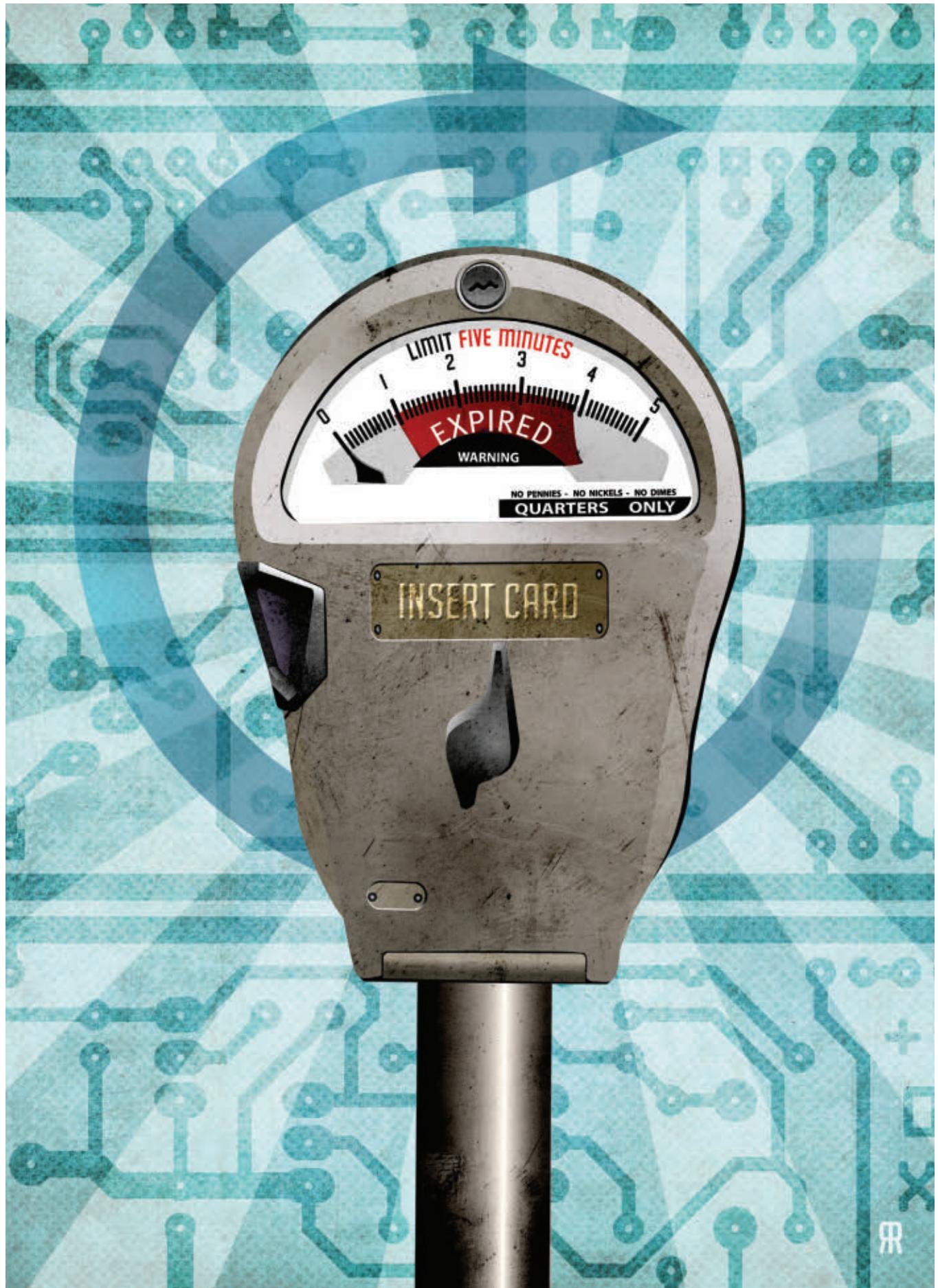


ILLUSTRATION BY RAFAEL RICOY

latency, transfer bandwidth, spatial density, power consumption, and cooling costs.¹³ Table 1 and some derived metrics in Table 2 illustrate this point (all metrics derived on 4/11/2007 from dramexchange.com, dvnation.com, buy.com, seagate.com, and samsung.com).

Given the number of CPU instructions possible during the time required for one disk I/O has steadily increased, an intermediate memory in the storage hierarchy is desirable. Flash memory seems to be a highly probable candidate, as has been observed many times by now.

Many architecture details remain to be worked out. For example, in the hardware architecture, will flash memory be accessible via a DIMM slot, a SATA (serial ATA) disk interface, or yet another hardware interface? Given the effort and delay in defining a new hardware interface, adaptations of existing interfaces are likely.

A major question is whether flash memory is considered a special part of either main memory or persistent storage. Asked differently: if a system includes 1GB traditional RAM, 8GB flash memory, and 250GB traditional disk, does the software treat it as

250GB of persistent storage and a 9GB buffer pool, or as 258GB of persistent storage and a 1GB buffer pool? The second goal of this article is to answer this question and, in fact, to argue for different answers in file systems and database systems.

Many design decisions depend on the answer to this question. For example, if flash memory is part of the buffer pool, pages must be considered “dirty” if their contents differ from the equivalent page in persistent storage. Synchronizing the file system or checkpointing a database must force disk writes in those cases. If flash memory is part of persistent storage, these write operations are not required.

Designers of operating systems and file systems will want to use flash memory as an extended buffer pool (extended RAM), whereas database systems will benefit from flash memory as an extended disk (extended persistent storage). Multiple aspects of file systems and database systems consistently favor these two designs. Presenting the case for these designs is the third goal of this article.

Finally, the characteristics of flash memory suggest some substantial

differences in the management of B-tree pages and their allocation. Beyond optimization of page sizes, B-trees can use different units of I/O for flash memory and disks. These page sizes lead to two new five-minute rules. Introducing these two new rules is the fourth goal of this article.

Assumptions

Forward-looking research relies on many assumptions. This section lists the assumptions that led to the conclusions put forth in this article. Some of these assumptions are fairly basic, whereas others are more speculative.

One assumption is that file systems and database systems assign the same data to the flash memory between RAM and the disk drive. Both software systems favor pages with some probability that they will be touched in the future but not with sufficient probability to warrant keeping them in RAM. The estimation and administration of such probabilities follows the usual lines, such as LRU (least recently used).

We assume that the administration of such information uses data structures in RAM, even for pages whose contents have been removed from RAM to flash memory. For example, the LRU chain in a file system’s buffer pool might cover both RAM and flash memory, or there might be two separate LRU chains. A page is loaded into RAM and inserted at the head of the first chain when it is needed by an application. When it reaches the tail of the first chain, the page is moved to flash memory and its descriptor to the head of the second LRU chain. When it reaches the tail of the second chain, the page is moved to disk and removed from the LRU chain. Other replacement algorithms would work *mutatis mutandis*.

Such fine-grained LRU replacement of individual pages is in contrast to assigning entire files, directories, tables, or databases to different storage units. It seems that page replacement is the appropriate granularity in buffer pools. Moreover, proven methods exist for loading and replacing buffer-pool contents entirely automatically, with no assistance from tuning tools or directives by users or administrators needed. An extended buffer pool in

Table 1: Prices and performance of flash and disks.

RAM	Flash disk	SATA disk
Price and capacity	\$3 for 8×64Mbit	\$999 for 32GB
Access latency	0.1ms	12ms average
Transfer bandwidth	66MB/s API	300MB/s API
Active power	1W	10W
Idle power	0.1W	8W
Sleep power	0.1W	1W

Table 2: Relative costs for flash memory and disks.

NAND Flash	SATA disk
Price and capacity	\$999 for 32GB
Price per GB	\$31.20
Time to read a 4KB page	0.16ms
4KB reads per second	6,200
Price per 4KB read per second	\$0.16
Time to read a 256KB page	3.98ms
256KB reads per second	250
Price per 256KB read per second	\$3.99
	\$1.03

flash memory should exploit the same methods as a traditional buffer pool. For truly comparable and competitive performance and administration costs, a similar approach seems advisable when flash memory is used as an extended disk.

File systems. Our research assumed a fairly traditional file system. Although many file systems differ from this model, most still generally follow it.

In our traditional system, each file is a large byte stream. Files are often read in their entirety, their contents manipulated in memory, and the entire file replaced if it is updated. Archiving, version retention, hierarchical storage management, data movement using removable media, among others, all seem to follow this model as well.

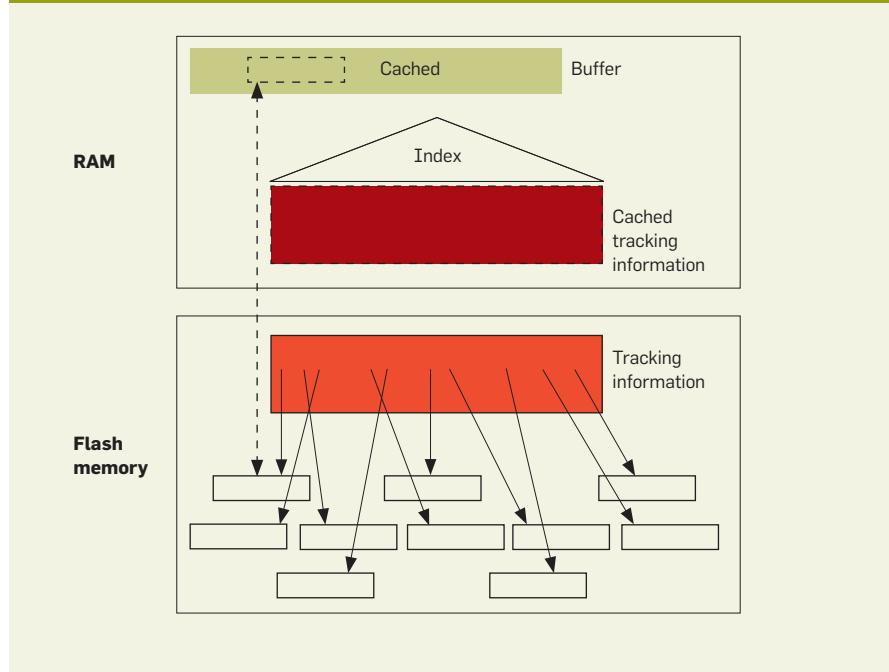
Based on this model, space allocation on disk attempts to use contiguous disk blocks for each file. Metadata is limited to directories, a few standard tags such as a creation time, and data structures for space management.

Consistency of these on-disk data structures is achieved by careful write ordering, fairly quick write-back of updated data blocks, and expensive file-system checks after any less-than-perfect shutdown or media removal. In other words, we assume the absence of transactional guarantees and transactional logging, at least for file contents. If log-based recovery is supported for file contents such as individual pages or records within pages, then a number of the arguments presented here need to be revisited.

Database systems. We assume fairly traditional database systems with B-tree indexes as the workhorse storage structure. Similar tree structures capture not only traditional clustered and nonclustered indexes, but also bitmap indexes, columnar storage, contents indexes, XML indexes, catalogs (metadata), and allocation data structures.

With respect to transactional guarantees, we assume traditional write-ahead logging of both contents changes (such as inserting or deleting a record) and structural changes (such as splitting B-tree nodes). Efficient log-based recovery after failures is enabled by checkpoints that force dirty data from the buffer pool to persistent storage.

Figure 1: Caching and indexing page locations.



Variations such as “second-chance” or fuzzy checkpoints fit within our assumptions. In addition, nonlogged (allocation-only logged) execution is permitted for some operations such as index creation. These operations require appropriate write ordering and a “force” buffer pool policy.¹⁸

Flash memory. Hardware and device drivers are assumed to hide many implementation details such as the specific hardware interface to flash memory. For example, flash memory might be mounted on the computer’s motherboard, a DIMM slot, a PCI board, or within a standard disk enclosure. In all cases, DMA transfers (or something better) are assumed between RAM and flash memory. Moreover, we assume there is either efficient DMA data transfer between flash and disk or a transfer buffer in RAM. The size of such a transfer buffer should be, in a first approximation, about equal to the product of transfer bandwidth and disk latency. If it is desirable that disk writes should never delay disk reads, the increased write-behind latency must be included in the calculation.

Another assumption is that transfer bandwidths of flash memory and disk are comparable. While flash write bandwidth has lagged behind read bandwidth, some products claim a difference of less than a factor of two

(for example, Samsung’s Flash-based solid-state disk used in Table 1). If necessary, the transfer bandwidth can be increased by using array arrangements, as is well known for disk drives; even redundant arrangement of flash memory may prove advantageous in some cases.⁶

Since the reliability of current NAND flash suffers after 100,000–1,000,000 erase-and-write cycles, we assume that some mechanisms for *wear leveling* are provided. These mechanisms ensure that all pages or blocks of pages are written similarly often. It is important to recognize the similarity between wear-leveling algorithms and log-structured file systems,^{22, 27} although the former also move stable, unchanged data such that their locations can absorb some of the erase-and-write cycles.

Note that traditional disk drives do not support more write operations, albeit for different reasons. For example, six years of continuous and sustained writing at 100Mbps overwrites an entire 250GB disk fewer than 80,000 times. In other words, assuming that a log-structured file system is appropriate for RAID-5 or RAID-6 arrays, the reliability of current flash seems comparable. Similarly, overwriting a 32GB flash disk 100,000 times with a sustained average bandwidth of 30Mbps takes about 3.5 years.

In addition to wear leveling, we assume that an asynchronous agent moves stale data from flash memory to disk and immediately erases the freed-up space in flash memory to prepare it for write operations without further delay. This activity also has an immediate equivalence in log-structured file systems—namely, the cleanup activity that prepares space for future log writing. The difference is that disk contents must merely be moved, whereas flash contents must also be erased before the next write operation at that location.

In either file systems or database systems, we assume separate mechanisms for page tracking and page replacement. A traditional buffer pool, for example, provides both, but it uses two different data structures for these two purposes. The standard design relies on an LRU list for page replacement and on a hash table for tracking pages (that is, which pages are present in the buffer pool and in which buffer frames). Alternative algorithms and data structures also separate page tracking and replacement management.

The data structures for the replacement algorithm are assumed to be small and have high traffic and are therefore kept in RAM. We also assume that page tracking must be as persistent as the data, including free-space information. Thus, a buffer pool's hash table is reinitialized during a system reboot, but tracking information for pages on a persistent store such as a disk must be stored with the data. The tracking information may well be cached in RAM while a volume is active, but any changes must be logged and written back to permanent storage. The index required to find the current location of a page may exist only in RAM, being reconstructed every time a volume is opened and the tracking information loaded into the cache in RAM.

As previously mentioned, we assume page replacement on demand. In addition, automatic policies and mechanisms may exist for prefetch, read-ahead, and write-behind.

Based on these considerations, we assume the contents of flash memory are pretty much the same, whether the flash memory extends the buffer pool

or the disk. The central question is therefore not what to keep in cache but how to manage flash-memory contents and its lifetime.

In database systems, flash memory can also be used for recovery logs, because its short access times permit very fast transaction commit. However, limitations in write bandwidth discourage such use. Perhaps systems with dual logs can combine low latency and high bandwidth, one log on a traditional disk and one log on an array of flash chips, with a slightly optimistic policy to consider a transaction committed as soon as the write operation on flash is complete.

Other hardware. In all cases, RAM is assumed to be a substantial size, although probably less than flash memory or disk. The relative sizes should be governed by the five-minute rule.¹⁵ Note that, despite similar transfer bandwidth, the short access latency of flash memory compared with disk results in surprising retention times for data in RAM.

Finally, we assume sufficient processing bandwidth as provided by modern many-core processors. Moreover, forthcoming transactional memory (in hardware and in the software runtime system) is expected to permit highly concurrent maintenance of complex data structures. For example, page replacement heuristics might use priority queues rather than bitmaps or linked lists. Similarly, advanced lock management might benefit from more complex data structures. Nonetheless, we neither assume nor require data structures more complex than those already in common use for page replacement and location tracking.

The Five-Minute Rule

If flash memory is introduced as an

intermediate level in the memory hierarchy, relative sizing of memory levels demands renewed consideration.

Tuning can be based on purchasing cost, total cost of ownership, power, mean time to failure, mean time to data loss, or a combination of metrics. Following Gray and Putzolu,¹⁵ this article focuses on purchasing cost. Other metrics and appropriate formulas to determine relative sizes can be derived similarly (for example, by replacing dollar costs with energy use for caching and moving data).

Gray and Putzolu introduced the following formula:^{14, 15}

$$\text{BreakEvenInterval in Seconds} = (\text{PagesPerMBofRAM} / \text{AccessesPerSecondPerDisk}) \times (\text{PricePerDiskDrive} / \text{PricePerMBofRAM}).$$

It is derived using formulas for the cost of RAM to hold a page in the buffer pool and the cost of a (fractional) disk to perform I/O every time a page is needed, equating these two costs, and solving the equation for the interval between accesses.

Assuming modern RAM, a disk drive using 4KB pages, and the values from Table 1 and Table 2, this produces

$$(256/83) \times (\$80/\$0.047) = 5,248 \text{ seconds} \\ \approx 90 \text{ minutes} = 1\frac{1}{2} \text{ hours}$$

(The “=” sign often indicates rounding in this article.)

This compares with two minutes (for 4KB pages) 20 years ago. If there is a surprise in this change, it is that the break-even interval has grown by less than two orders of magnitude. Recall that RAM was estimated in 1987 at about \$5,000 per megabyte, whereas the 2007 cost is about \$0.05 per megabyte, a difference of five orders of magnitude. On the other

Table 3: Break-even intervals [seconds].

Page size	1KB	4KB	16KB	65KB	256KB
RAM-SATA	20,978	5,248	1,316	334	88
RAM-flash	2,513	876	467	365	339
Flash-SATA	32,253	8,070	2,024	513	135
RAM-\$400	1,006	351	187	146	136
\$400-SATA	80,553	20,155	5,056	1,281	337

hand, disk prices have also tumbled (\$15,000 per disk in 1987), and disk latency and bandwidth have improved considerably (from 15 accesses per second to about 100 on consumer disks and 200 on high-performance enterprise disks).

For RAM and flash disks of 32GB, the break-even interval is

$$(256 / 6,200) \times (\$999 / \$0.047) = 876 \text{ seconds} \approx 15 \text{ minutes}$$

If the 2007 price for flash disks includes a “novelty premium” and comes down closer to the price of raw flash memory—say, to \$400 (a price also anticipated by Gray and Fitzgerald¹³)—then the break-even interval is 351 seconds \approx 6 minutes.

An important consequence is that in systems tuned using economic considerations, turnover in RAM is about 15 times faster (90 minutes / 6 minutes) if flash memory rather than a traditional disk is the next level in the storage hierarchy. Much less RAM is required, resulting in lower costs for purchase, power, and cooling.

Perhaps most interesting, applying the same formula to flash and disk results in the following:

$$(256 / 83) \times (\$80 / \$0.03) = 8,070 \text{ seconds} \approx 2\frac{1}{4} \text{ hours}$$

Thus, all active data will remain in RAM and flash memory.

Without a doubt, two hours is longer than any common checkpoint interval, which implies that dirty pages in flash are forced to disk not by page replacement but by checkpoints. Pages that are updated frequently must be written much more frequently (because of checkpoints) than is optimal based on Gray and Putzolu’s formula.

In 1987, Gray and Putzolu speculated 20 years into the future and anticipated a “five-hour rule” for RAM and disks. For 1KB records, prices and specifications typical in 2007 suggest 20,978 seconds, or just under six hours. Their prediction was amazingly accurate.

All break-even intervals are different for larger page sizes (64KB or even 256KB). Table 3 shows the break-even intervals, including those just cited, for a variety of page sizes and

Flash memory falls between traditional RAM and persistent mass storage based on rotating disks in terms of acquisition cost, access latency, transfer bandwidth, spatial density, power consumption, and cooling costs.

combinations of storage technologies. (“\$400” stands for a 32GB NAND flash drive available in the future rather than for \$999 in 2007; in fact, 32GB SLC SATA drives are available at retail for \$400 in 2009.)

The old five-minute rule for RAM and disk now applies to 64KB page sizes (334 seconds). Five minutes had been the approximate break-even interval for 1KB in 1987¹⁵ and for 8KB in 1997.¹⁴ This trend reflects the different rates of improvement in disk-access latency and transfer bandwidth.

The five-minute break-even interval also applies to RAM and the expensive flash memory of 2007 for page sizes of 64KB and above (365 seconds and 339 seconds). As the price premium for flash memory decreases, so does the break-even interval (146 seconds and 136 seconds).

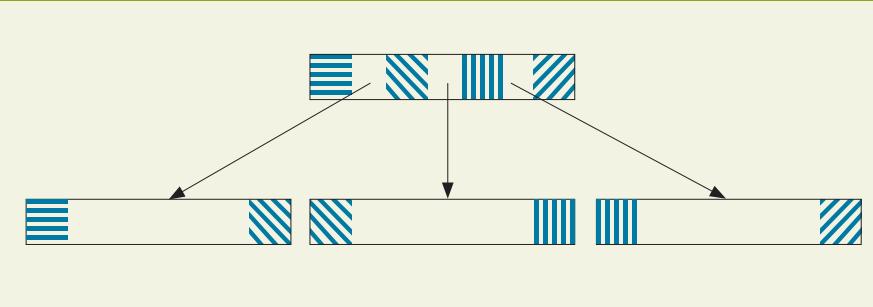
Two new five-minute rules are indicated with values in ***bold italics*** in Table 3. We will come back to this table and these rules in the discussion on optimal node sizes for B-tree indexes.

Page Movement

In addition to I/O to and from RAM, a three-level memory hierarchy also requires data movement between flash memory and disk storage.

The pure mechanism for moving pages can be realized in hardware (for example, by DMA transfer), or it might require an indirect transfer via RAM. The former case promises better performance, whereas the latter design can be realized entirely in software without novel hardware. On the other hand, hybrid disk manufacturers might have cost-effective hardware implementations already available.

The policy for page movement is governed or derived from demand-paging and LRU replacement. As mentioned earlier, replacement policies in both file systems and database systems may rely on LRU and can be implemented with appropriate data structures in RAM. As with buffer management in RAM, there may be differences resulting from prefetch, read-ahead, and write-behind. In database systems these may be directed by hints from the query execution layer, whereas file systems must detect page-access patterns

Figure 2: A write-optimized B-tree with fence keys instead of neighbor pointers.

and worthwhile read-ahead actions without the benefit of such hints.

If flash memory is part of the persistent storage, page movement between flash memory and disk is similar to page movement during defragmentation, both in file systems and database systems. The most significant difference is how page movement and current page locations are tracked in these two kinds of systems.

Tracking Page Locations

The mechanisms for tracking page locations are quite different in file systems and database systems. In file systems, pointer pages keep track of data pages or runs of contiguous data pages. Moving an individual page may require breaking up a run. It always requires updating and then writing a pointer page.

In database systems, most data is stored in B-tree indexes, including clustered (primary, nonredundant) and nonclustered (secondary, redundant) indexes on tables, materialized views, and database catalogs. Bitmap indexes, columnar storage, and master-detail clustering can be readily and efficiently represented in B-trees.¹² Tree structures derived from B-trees are also used for *blobs* (binary large objects) and are similar to the storage structures of some file systems.^{5,25}

For B-trees, moving an individual page can be very expensive or very cheap. The most efficient mechanisms are usually found in utilities for defragmentation or reorganization. Cost or efficiency results from two aspects of B-tree implementation—namely, maintenance of neighbor pointers, and logging for recovery.

First, if physical neighbor pointers are maintained in each B-tree page, moving a single page requires updating two neighbors in addition to the

parent node. If the neighbor pointers are logical using *fence keys*, only the parent page requires an update during a page movement.¹⁰ Figure 2 shows such a B-tree, with neighbor pointers replaced by copies of the separator keys propagated to the parent node during leaf splits. If the parent page is in memory, perhaps even pinned in the buffer pool, recording the new location is rather like updating an in-memory indirection array. The pointer change in the parent page is logged in the recovery log, but there is no need to force the log immediately to stable storage because this change is merely a structural change, not a database contents change.

Second, database systems log changes in the physical database, and in the extreme case both the deleted page image and the newly created page image are logged. Thus, an inefficient implementation fills two log pages whenever a single data page moves from one location to another. A more efficient implementation logs only allocation actions and delays deallocation of the old page image until the new image is safely written in its intended location.¹⁰ In other words, moving a page from one location (for example, on persistent flash memory) to another (for example, on disk) requires only a few bytes in the database recovery log.

The difference between traditional file systems and database systems is the efficiency of updates enabled by the recovery log. In a file system, the new page location must be saved as soon as possible by writing a new image of the pointer page. In a database system, only a single log record or a few short log records must be added to the log buffer. Thus, the overhead for a page movement in a file system is writing an entire pointer

page using a random access, whereas a database system adds a log record of a few dozen bytes to the log buffer that will eventually be written using large sequential write operations.

If a file system uses flash memory as persistent storage, moving a page between a flash memory location and an on-disk location adds substantial overhead. Thus, most file-system designers will probably prefer flash memory as an extension to the buffer pool rather than as an extension of the disk, thus avoiding this overhead.

A database system, however, has built-in mechanisms that can easily track page movements. These mechanisms are inherent in the “workhorse” data structure, B-tree indexes. Compared with file systems, these mechanisms permit efficient page movement, each one requiring only a fraction of a sequential write (in the recovery log) rather than a full random write.

Moreover, the database mechanisms are reliable. Should a failure occur during a page movement, database recovery is driven by the recovery log, whereas a traditional file system requires checking the entire volume during reboot.

Checkpoint Processing

To ensure fast recovery after a system failure, database systems use checkpoints. Their effect is that recovery needs to consider database activity only from the most recent checkpoint, plus some limited activity explicitly indicated in the checkpoint information. The main effort is writing dirty pages from the buffer pool to persistent storage.

If pages in flash memory are part of the buffer pool, dirty pages must be written to disk during database checkpoints. Common checkpoint intervals are measured in seconds or minutes. Alternatively, if checkpoints are not truly points but intervals, it is reasonable to flush pages and perform checkpoint activities continuously, starting the next checkpoint as soon as one finishes. With flash memory as part of the buffer pool, many writes to flash memory require a write to disk soon thereafter as part of checkpoint processing, and flash memory as the intermediate level in the memory hierarchy fails to absorb write activity. Recall, this effect may be exacerbated

if RAM is kept small because of the presence of flash memory.

If, on the other hand, flash memory is considered persistent storage, writing to flash memory is sufficient. Write-through to disk is required only as part of page replacement (such as, when a page's usage suggests placement on disk rather than in flash memory). Thus, checkpoints do not incur the cost of moving data from flash memory to disk.

Checkpoints might even be faster in systems with flash memory because dirty pages in RAM need to be written merely to flash memory, not to disk. Given the very fast random access in flash memory relative to disk drives, this difference might speed up checkpoints significantly.

To summarize, database systems benefit if flash memory is treated as part of the system's persistent storage. In contrast, traditional file systems do not have systemwide checkpoints that flush the recovery log and any dirty data from the buffer pool. Instead, they rely on carefully writing modified file-system pages because of the lack of a recovery log protecting file contents.

Page Sizes

In addition to tuning based on the five-minute rule, another optimization based on access performance is sizing of B-tree nodes. The optimal page size minimizes the time spent on I/O during a root-to-leaf search. It balances a short I/O (that is, the desire for small pages) with a high reduction in remaining search space (that is, the desire for large pages).

Assuming binary search within each B-tree node, the reduction in remaining search space is measured by the logarithm of the number of records within each node. This measure was called a node's *utility* in our earlier work.¹⁴ This optimization is essentially equivalent to one described in the original research on B-trees.³

Table 4 illustrates this optimization for 20-byte records (typical with prefix and suffix truncation⁴) and for nodes filled at about 70%.

Not surprisingly, the optimal node size for B-tree indexes on modern high-bandwidth disks is much larger than the page sizes in traditional database systems. With those disks,

the access time dominates for all small page sizes, such that additional byte transfer and thus additional utility are almost free.

B-tree nodes of 256KB are near optimal. For those, Table 3 indicates a break-even time for RAM and disk of 88 seconds. For a \$400 flash disk and a traditional rotating hard disk, Table 4 indicates 337 seconds or just over five minutes. This is the first of the two new five-minute rules.

Table 5 illustrates the same calculations for B-trees on flash memory. Because there is no mechanical seeking or rotation, transfer time dominates access time even for small pages. The optimal page size for B-trees on flash memory is 2KB, much smaller than for traditional disk drives. In Table 3, the break-even interval for 4KB pages

is 351 seconds. This is the second new five-minute rule.

The implication of two different optimal page sizes is that any uniform node size for B-trees on flash memory and traditional rotating hard disks is suboptimal. Optimizing page sizes for both media requires a change in buffer management, space allocation, and some of the B-tree logic.

Fortunately, Patrick O'Neil of the University of Massachusetts already designed a space allocation scheme for B-trees in which neighboring leaf nodes usually reside within the same contiguous extent of pages.²³ When a new page is needed for a node split, another page within the same extent is allocated. When an extent overflows, half its pages are moved to a newly allocated extent. In other words, the

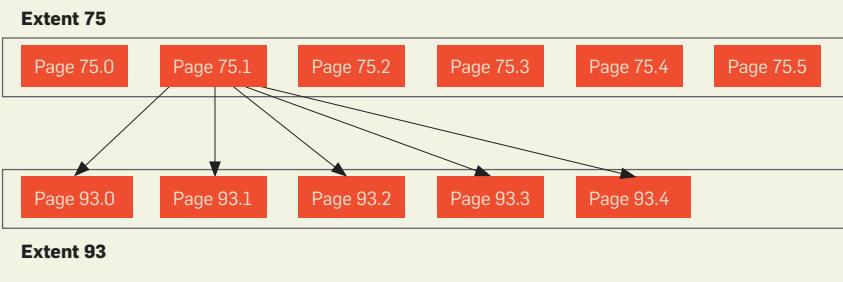
Table 4: Page utility for B-tree nodes on disk.

Page size	Records per page	Node utility	Access time	Utility/time
4KB	140	7	12.0ms	0.58
16KB	560	0	12.1ms	0.75
64KB	2,240	11	12.2ms	0.90
128KB	4,480	12	12.4ms	0.97
256KB	8,960	13	12.9ms	1.01
512KB	17,920	14	13.7ms	1.02
1MB	35,840	15	15.4ms	0.97

Table 5: Page utility for B-tree nodes on flash memory.

Page size	Records per page	Node utility	Access time	Utility/time
1KB	35	5	0.11ms	43.4
2KB	70	6	0.13ms	46.1
4KB	140	7	0.16ms	43.6
8KB	280	8	0.22ms	36.2
16KB	560	9	0.34ms	26.3
64KB	2,240	11	1.07ms	10.3

Figure 3: Pages and extents in an SB-tree.



“split in half when full” logic of B-trees is applied not only to pages containing records, but also to contiguous disk extents containing pages.

Using O’Neil’s SB-trees (*S* meaning *sequential*), 256KB extents can be the units of transfer between flash memory and disk, whereas 4KB pages can be the unit of transfer between RAM and flash memory. Figure 3 shows pages within two extents. Child pointers in a B-tree (also shown) refer to individual pages. If multiple neighboring child pointers refer to neighboring pages (as shown), the pointer values can be represented compactly with run-length encoding applied not to a set of duplicate key values but to a series of values with constant increments. For example, the five child pointers in extent 75.1 in Figure 3 can be represented by the page identifier 93.0 and the counter 5.

Similar notions of self-similar B-trees have also been proposed for higher levels in the memory hierarchy, for example, in the form of B-trees of cache lines for the indirection vector within a large page.¹⁹ Given that there are at least three levels of B-trees and three node sizes now (cache lines, flash memory pages, and disk pages), research into cache-oblivious B-trees² might be promising.

Database-Query Processing

Self-similar designs apply both to data structures such as B-trees and to algorithms. For example, sort algorithms already use algorithms similar to traditional external merge sorts in multiple ways—to merge runs not only on disk but also in memory, where the initial runs are sized to limit run creation to the CPU cache.^{11,21}

The same technique might be applied three times instead of twice: first, cache-size runs in memory are merged into memory-size runs in memory; second, in larger sort operations, memory-size runs in flash memory are merged into runs on disk; and third, runs on disk are merged to form the final sorted result. Read-ahead, forecasting, write-behind, and page sizes all deserve a new look in a multilevel memory hierarchy consisting of cache, RAM, flash memory, and traditional disk drives. These page sizes can then inform the break-even calculation for page retention versus I/O and thus



The 20-year-old five-minute rule for RAM and disks still holds, but for ever-larger disk pages. Moreover, it should be augmented by two new five-minute rules: one for small pages moving between RAM and flash memory and one for large pages moving between flash memory and traditional disks.



guide the optimal capacities at each level of the memory hierarchy.

We can surmise that a variation of this sort algorithm will be not only fast but also energy efficient. While energy efficiency has always been crucial for battery-powered devices, research into energy-efficient query processing on server machines is only now beginning.²⁴ For example, for both flash memory and disks, energy-optimal page sizes might well differ from performance-optimal page sizes.

The I/O pattern of an external merge sort is similar (albeit in the opposite direction) to the I/O pattern of an external distribution sort. Figure 4 illustrates how merging combines many small files into a large file, with many seek operations in the small files as demanded by the merge logic, and how partitioning divides a single large file into many small files, with many seek operations in the small files as demanded by the partitioning function. The I/O pattern of a distribution sort is equal to that of partitioning during hash join and hash aggregation.⁸ All of these algorithms require reevaluation and redesign in a three-level memory hierarchy, or even a four-level hierarchy if CPU caches are also considered.²⁶

Flash memory with its very fast access times may well revive interest in index-based query execution.^{7,9} Instead of large scans and memory-intensive operations such as sorting and hash join, fast accesses to index pages shift the break-even point toward index-to-index navigation. For example, assume a table with 100 million rows of 100 bytes and table scans at 100MB per second. A table scan takes 100 seconds. Searching a secondary index requires fetching individual rows from the table. If the table is stored on a traditional disk, then a period of 100 seconds permits fetching about 10,000 records. If more than 10,000 rows satisfy the query predicate, then the table scan is faster. If, however, the table is stored on a flash device, 100 seconds will permit fetching about 500,000 records. Thus, flash storage shifts the break-even point between table scan and index search from 10,000 to 500,000 rows satisfying the query predicate, and many more query execution plans will rely on index-to-index navigation rather than large scans and hash joins.

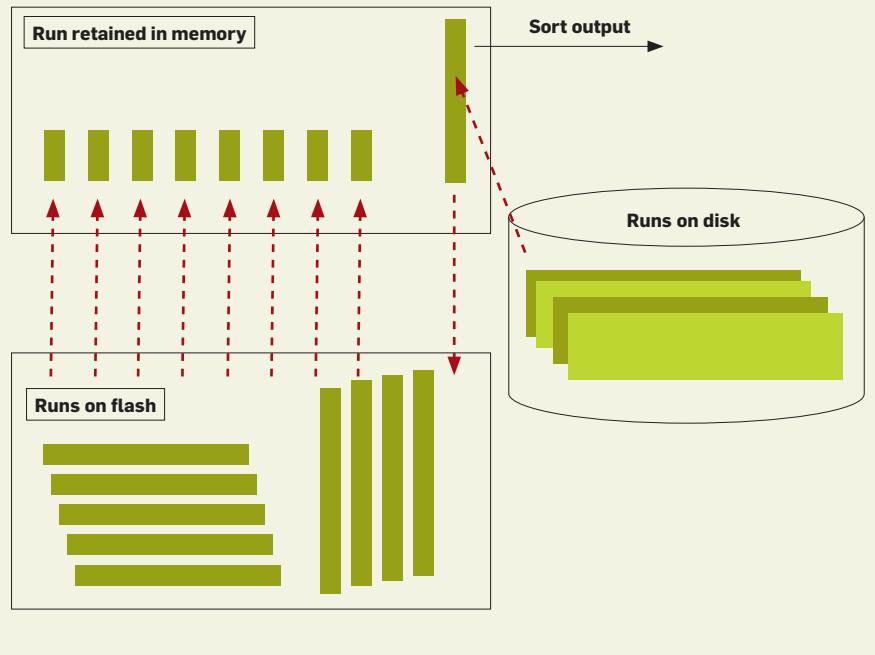
Multiple secondary indexes for a single table can be exploited into index intersection, index joins, among others. Fast access to individual pages and records also benefits those query execution plans. Like secondary indexes, column stores or more generally vertical partitioning also require fetching records from multiple places to assemble complete rows. Thus, as seen in the example of database query processing, using flash memory in addition to or even as replacement of traditional disks not only forces reevaluation of optimal use of the hardware but also means some substantial software changes.

Record and Object Caches

Page sizes in database systems have grown over the years, although not as fast as disk-transfer bandwidth. On the other hand, small pages require less buffer-pool space for each root-to-leaf search. For example, consider an index with 20 million entries. With index pages of 128KB and 4,500 records, a root-to-leaf search requires two nodes and thus 256KB in the buffer pool, although half of that (the root node) can probably be shared with other transactions. With 8KB index pages and 280 records per page, a root-to-leaf search requires three nodes or 24KB in the buffer pool, or one order of magnitude less.

In traditional database architecture, the default page size is a compromise between efficient index search (using large B-tree nodes as previously discussed here and in the original B-tree papers³) and moderate buffer-pool requirements for each index search. Nonetheless, the previous example requires 24KB in the buffer pool for finding a record of perhaps only 20 bytes,

Figure 5: Graceful degradation with multiple levels in the memory hierarchy.



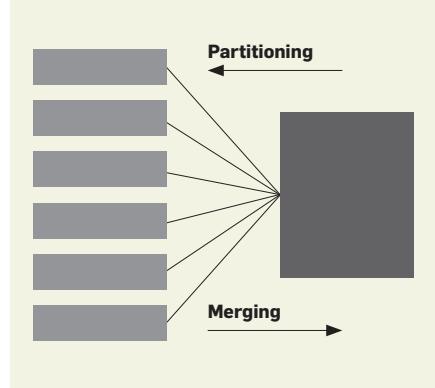
and it requires 8KB of the buffer pool for retaining these 20 bytes in memory. An alternative design uses large on-disk pages and a record cache that serves applications, because record caches minimize memory needs yet provide the desired data retention. In-memory databases represent a specific form of record caches when used as front ends for traditional disk-based databases.

The introduction of flash memory with its fast access latency and its small optimal page size may render record caches obsolete. With the large on-disk pages in flash memory and only small pages in the in-memory buffer pool, the desired compromise can be achieved without the need for two separate data structures (such as, a transacted B-tree and a separate record cache).

ware techniques with the highest impact on energy consumption. Note that traditional database-query processing relies on asynchronous I/O to reduce response times; if the primary cost metric for query processing is energy consumption, asynchronous I/O has no advantage over synchronous I/O.

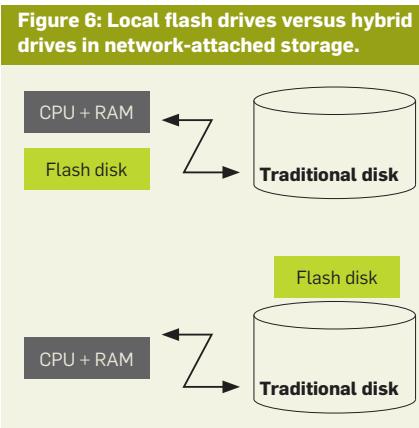
Second, the five-minute rule applies to permanent data and its management in a buffer pool. The optimal retention time for temporary data such as run files in sorting and overflow files in hash join and hash aggregation may be different. For sorting, as for B-tree searches, the goal should be to maximize the number of comparisons per unit of I/O time or per unit of energy spent on I/O. Our initial research and algorithm design has focused on algorithms with graceful degradation in sorting and for hybrid hash join (that is, spilling memory contents to flash only when and as much as truly required, and similarly spilling flash contents to disk only when and as much as truly required). The different optimal page sizes can be exploited to achieve very high effective merge fan-in and partitioning fan-out with relatively little main memory. Figure 5 shows the final merge step—very large runs on disk use large pages that are buffered in flash memory (shown as vertical boxes), a few small runs have remained in flash

Figure 4: Merging and partitioning files.



Future Work

Several directions for future research suggest themselves. First, while the analyses in this article focused on purchasing costs, a consideration of other costs could capture the total cost of ownership. A focus on energy consumption, for example, could lead to different break-even points or even entirely different conclusions. Along with CPU scheduling, algorithms for staging data in the memory hierarchy—including buffer-pool replacement and asynchronous I/O—may be the soft-



and never were merged to form very large runs on disk (shown as horizontal boxes), and the available RAM is used to merge a very large number of runs exploiting the small page size optimal for flash devices.

Third, Gray and Putzolu offered further rules of thumb, such as the 10-byte rule for trading memory and CPU power. These rules also warrant revisiting for both costs and energy. Compared with 1987, the most fundamental change may be that CPU power should be measured not in instructions but in cache line replacements. Trading off space and time seems like a new problem in an environment with multiple levels in the memory hierarchy. A modern memory hierarchy might be very deep: multiple levels of CPU caches, main memory (possibly in a NUMA design), flash devices, and finally performance-optimized “enterprise” disks and capacity-optimized “consumer” disks. The lower levels may rely on various software techniques with different trade-offs between performance and reliability, such as striping, mirroring, single-redundancy RAID-5, dual-redundancy RAID-6, log-structured file systems, and write-optimized B-trees.

Fourth, what are the best data movement policies? One extreme is a database administrator explicitly moving entire files, tables, or indexes between flash memory and traditional disk. Another extreme is automatic movement of individual pages, controlled by a replacement policy such as LRU. Intermediate policies may focus on the roles of individual pages within a database or on the current query-processing activity. For example, all catalog pages may be moved as a

unit after schema changes to facilitate fast recompilation of all cached query execution plans, and all relevant upper B-tree levels may be prefetched and cached in RAM or in flash memory during execution of query plans relying on index-to-index navigation. The variety of possibilities may overwhelm automatic policies and may require hints or directives from applications or database software.

Fifth, what are the secondary and tertiary effects of introducing flash memory into the memory hierarchy of a database server? For example, short access times permit a lower multi-programming level, because only short I/O operations must be hidden by asynchronous I/O and context switching. A lower multi-programming level in turn may reduce contention for memory in sort and hash operations, locks (concurrency control for database contents), and latches (concurrency control for in-memory data structures). Should this effect prove significant, the effort and complexity of using a fine granularity of locking may be reduced. Page-level concurrency control may also be sufficient simply as a result of small page sizes. Similarly, in-page data structures may require less optimization, although some techniques may apply to small pages (optimized for flash) within large pages (optimized for disks)—for example, clustering records versus clustering fields.¹

Sixth, will hardware architecture considerations invalidate some of the findings and conclusions of this article? For example, disks are currently separated from the main processors (for example, in network-attached storage or storage-area networks). Will flash devices be placed with the main processors? If so, is it still a good idea to use flash devices as extended disk rather than extended buffer pool? Figure 6 shows two of these alternatives. In the top arrangement, questions arise about the scope and effectiveness of centralized storage management, the granularity of failures and replacement, and so on, whereas many of these questions have much more obvious answers in the bottom arrangement.

Seventh, how will flash memory affect in-memory database systems? Will they become more scalable,

affordable, and popular based on memory inexpensively extended with flash memory rather than RAM? Will they become less popular as a result of very fast traditional database systems using flash memory instead of (or in addition to) disks? Can a traditional code base using flash memory instead of traditional disks compete with a specialized in-memory database system in terms of performance, total cost of ownership, development and maintenance costs, or time to market of features and releases? What techniques in the buffer pool are required to achieve performance competitive with in-memory databases? For example, the upper levels of B-tree indexes can be pinned in the buffer pool and augmented with memory addresses of all child pages (or their buffer descriptors) also pinned in the buffer pool, and auxiliary structures may enable efficient interpolation search instead of binary search.

Finally, techniques similar to generational garbage collection may benefit storage hierarchies.²² Selective reclamation applies not only to unreachable in-memory objects but also to buffer-pool pages and favored locations on permanent storage. Such research also may provide guidance for log-structured file systems, wear leveling for flash memory, and write-optimized B-trees on RAID storage.

Conclusion

The 20-year-old five-minute rule for RAM and disks still holds, but for ever-larger disk pages. Moreover, it should be augmented by two new five-minute rules: one for small pages moving between RAM and flash memory and one for large pages moving between flash memory and traditional disks. For small pages moving between RAM and disk, Gray and Putzolu were amazingly accurate in predicting a five-hour break-even point two decades into the future.

Research into flash memory and its place in system architectures is urgent and important. Within a few years, flash memory will be used to fill the gap between traditional RAM and traditional disk drives in many operating systems, file systems, and database systems.

Flash memory can be used to extend

RAM or persistent storage. These models are called *extended buffer pool* and *extended disk* here. Both models may seem viable in operating systems, file systems, and in database systems. The different characteristics of each of these systems, however, will require different usage models.

In both models, contents of RAM and flash will be governed by LRU-like replacement algorithms that attempt to keep the most valuable pages in RAM and the least valuable pages on traditional disks. The linked list or other data structure implementing the replacement policy for flash memory will be maintained in RAM.

Operating systems and traditional file systems will use flash memory mostly as transient memory (for example, as a fast backup store for virtual memory and as a secondary file-system cache). Both of these applications fall into the extended buffer-pool model. During an orderly system shutdown, the flash memory contents must be written to persistent storage. During a system crash, however, the RAM-based description of flash-memory contents will be lost and must be reconstructed by a contents analysis similar to a traditional file-system check. Alternatively, flash-memory contents can be voided and reloaded on demand.

Database systems, on the other hand, will employ flash memory as persistent storage, using the extended disk model. The current contents will be described in persistent data structures, such as parent pages in B-tree indexes. Traditional durability mechanisms—in particular, logging and checkpoints—ensure consistency and efficient recovery after system crashes. Checkpoints and orderly system shutdowns have no need to write flash memory contents to disk, and the pre-shutdown of flash contents is required for a complete restart.

There are two reasons for these different usage models for flash memory. First, database systems rely on regular checkpoints during which dirty pages are flushed from the buffer pool to persistent storage. If a dirty page is moved from RAM to the extended buffer pool in flash memory, it creates substantial overhead during the next checkpoint. A free buffer must be found

in RAM, the page contents must be read from flash memory into RAM, and then the page must be written to disk. Adding such overhead to checkpoints is not attractive in database systems with frequent checkpoints. Operating systems and traditional file systems, on the other hand, do not rely on checkpoints and thus can exploit flash memory as an extended buffer pool.

Second, the principal persistent data structures of databases, B-tree indexes, provide precisely the mapping and location-tracking mechanisms needed to complement frequent page movement and replacement. Thus, tracking a data page when it moves between disk and flash relies on the same data structure maintained for efficient database search. In addition, avoiding indirection in locating a page also makes database searches as efficient as possible.

Finally, as the ratio of access latencies and transfer bandwidth is very different for flash memory and disks, different B-tree node sizes are optimal. O’Neil’s SB-tree exploits two different node sizes as needed in a multilevel storage hierarchy. The required inexpensive mechanisms for moving individual pages are the same as those required when moving pages between flash memory and disk.

Acknowledgments

This article is dedicated to Jim Gray, who suggested this research and helped the author and many others many times in many ways. Barb Peters, Lily Jow, Harumi Kuno, José Blakeley, Mehul Shah, the DaMoN 2007 reviewers, and particularly Harumi Kuno suggested multiple improvements after reading earlier versions of this work. □

References

1. Ailamaki, A., DeWitt, D.J. and Hill, M.D. Data page layouts for relational databases on deep memory hierarchies. *VLDB Journal* 11, 3 (2002), 198–215.
2. Bender, M.A., Demaine, E.D. and Farach-Colton, M. Cache-oblivious B-trees. *SIAM Journal on Computing* 35, 2 (2005), 341–358.
3. Bayer, R. and McCreight, E.M. Organization and maintenance of large ordered indexes. SIGFI-DET Workshop (1970), 107–141.
4. Bayer, R. and Uterauer, K. Prefix B-trees. *ACM Transactions on Database Systems* 2, 1 (1977), 11–26.
5. Carey, M.J., DeWitt, D.J., Richardson, J.E. and Shekita, E.J. Storage management in EXODUS. In *Object-Oriented Concepts, Databases, and Applications*. W. Kim and F. Lochovsky, Eds. ACM, N.Y., 1989, 341–369.
6. Chen, P.M., Lee, E.L., Gibson, G.A., Katz, R.H. and Patterson, D.A. 1994. RAID: high-performance, reliable secondary storage. *ACM Computing Surveys* 26(2): 145–185.
7. DeWitt, D.J., Naughton, J.F. and Burger, J. Nested loops revisited. *Parallel and Distributed Information Systems* (1993), 230–242.
8. Graefe, G. Query evaluation techniques for large databases. *ACM Computing Surveys* 25, 2 (1993), 73–170.
9. Graefe, G. Executing nested queries. *Database Systems for Business, Technology and Web* (2003), 58–77.
10. Graefe, G. Write-optimized B-trees. *VLDB Journal* (2004), 672–683.
11. Graefe, G. Implementing sorting in database systems. *ACM Computing Surveys* 38, 3 (2006), 69–106.
12. Graefe, G. Master-detail clustering using merged indexes. *Informatik-Forschung und Entwicklung*, 2007.
13. Gray, J. and Fitzgerald, B. 2007. Flash disk opportunity for server-applications; <http://research.microsoft.com/~gray/papers/FlashDiskPublic.doc>.
14. Gray, J., Graefe, G. 1997. The five-minute rule ten years later, and other computer storage rules of thumb. *SIGMOD Record* 26, 4 (1997), 63–68.
15. Gray, J. and Putzolu, G.R. The 5-minute rule for trading memory for disk accesses and the 10-byte rule for trading memory for CPU time. *SIGMOD Journal* (1987), 395–398.
16. Härdler, T. Implementing a generalized access path structure for a relational database system. *ACM Transactions on Database Systems* 3, 3 (1978), 285–298.
17. Hamilton, J. An architecture for modular data centers. In *Proceedings of the Conference on Innovative Data Systems Research*, 2007.
18. Härdler, T. and Reuter, A. Principles of transaction-oriented database recovery. *ACM Computing Surveys* 15, 4 (1983), 287–317.
19. Lomet, D.B. The evolution of effective B-tree page organization and techniques: a personal account. *SIGMOD Record* 30, 3, 64–69.
20. Larus, J.R. and Rajwar, R. *Transactional Memory. Synthesis Lectures on Computer Architecture*. Morgan & Claypool, 2007.
21. Nyberg, C., Barclay, T., Cvetanovic, Z., Gray, J. and Lomet, D.B. AlphaSort: A cache-sensitive parallel external sort. *VLDB Journal* (1995), 603–627.
22. Ousterhout, J.K. and Douglis, F. Beating the I/O bottleneck: A case for log-structured file systems. *Operating Systems Review* 23, 1 (1989), 11–28.
23. O’Neil, P.W. The SB-tree: An index-sequential structure for high-performance sequential access. *Acta Informatica* 29, 3 (1992), 241–265.
24. Rivoire, S., Shah, M., Ranganathan, P. and Kozyrakis, C. JouleSort: A balanced energy-efficiency benchmark. *SIGMOD Record*, 2007.
25. Stonebraker, M. Operating system support for database management. *Commun. ACM* 24, 7 (July 1981), 412–418.
26. Shatdal, A., Kant, C. and Naughton, J.F. Cache-conscious algorithms for relational query processing. *VLDB Journal* (1994), 510–521.
27. Woodhouse, D. JFFS: The Journaling Flash File System. Ottawa Linux Symposium, Red Hat Inc., 2001.

Related articles on queue.acm.org

Flash Storage Today

Adam Leventhal

<http://queue.acm.org/detail.cfm?id=1413262>

Flash Disk Opportunity for Server Applications

Jim Gray, Bob Fitzgerald

<http://queue.acm.org/detail.cfm?id=1413261>

Enterprise SSDs

Mark Moshayedi, Patrick Wilkison

<http://queue.acm.org/detail.cfm?id=1413263>

Goetz Graefe (Goetz.Graefe@HP.com) joined Hewlett-Packard Laboratories after seven years as an academic researcher and teacher followed by 12 years as a product architect and developer at Microsoft. He was recently named an HP Fellow. His Volcano research project was awarded the 10-year Test-of-Time Award at ACM SIGMOD 2000 for work on query execution.

An earlier version of this article was originally published in *Proceedings of the Third International Workshop on Data Management on New Hardware* (June 15, 2007), Beijing, China.

© 2009 ACM 0001-0782/09/0700 \$10.00

DOI:10.1145/1538788.1538806



Article development led by ACM Queue
queue.acm.org

The laws of physics and the Internet's routing infrastructure affect performance in a big way.

BY JONATHAN M. SMITH

Fighting Physics: A Tough Battle

OVER THE PAST several years, software-as-a-service (SaaS) has become an attractive option for companies looking to save money and simplify their computing infrastructures. SaaS is an interesting group of techniques for moving computing from the desktop to the cloud. However, as it grows in popularity, engineers should be aware of some of the fundamental limitations they face when developing these kinds of distributed applications—in particular, the finite speed of light.

Consider a company that wants to build a distributed application that does interprocess communication (IPC) over the long haul. The obvious advice is “just say no”—don’t do it. If you’re going far outside your local networking environment, the physics of distance and the speed of light, combined with the delays that come from the Internet’s routing infrastructure, tell us that it will be much too slow.

These concepts are not generally understood, however, and even when they are, they’re sometimes forgotten.

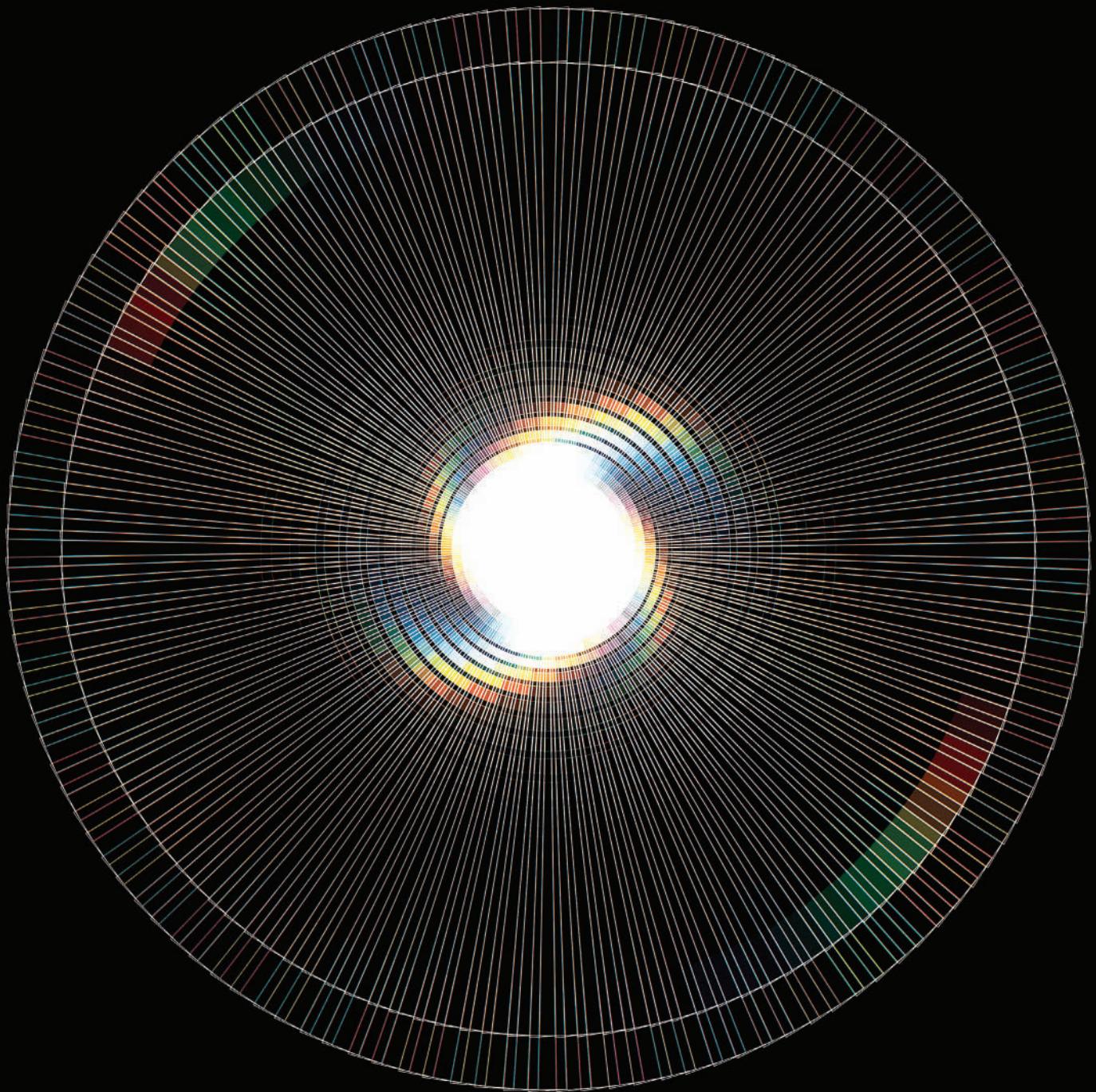
So, what are the basic principles related to speed of light and network hops that all software developers should be acquainted with? This article answers that question by first working out some quantitative preliminaries with an example, then moving on to the networking implications, and then covering applications. Finally, it provides some rules of thumb to keep in mind as applications and architectures evolve in reaction to new network capabilities and unchanging physics.

The Physics

The speed of light in a vacuum is exactly 299,792,458m/sec.² This is as fast as you can move a bit of data, and according to our current understanding of physics, it is a fundamental constraint of the universe in which we live. In fiber, the speed of light is 2.14×10^8 m/sec or about 70% of the speed of light in a vacuum. If a fiber were stretched in a straight line from New York to San Francisco, it would be about 4,125km long, and it would take about 19ms ($4,125 \div 214$) for light to make the one-way trip. Assuming an 8,250km length of fiber was used, you can just double this time to get an estimate for minimum round-trip time.

At first glance, 19ms might seem like a short time, certainly on a human scale. As computer scientists, however, we are usually concerned with a different time scale, that of the computer. Here we can calculate the 19ms in terms of instructions, the fundamental units of work for computers. As an example, we can use a 2003-vintage single-core machine: the Intel Pentium 4 Extreme Edition, which at a 3.2GHz clock rate was rated at 9,726MIPS: $9,726 \times 0.019$ is 184 million instructions—sufficient, for example, to search through or sort millions of names.

It is always important to keep in mind the purpose of computer networking is to interconnect computers, and that computers operate on very



short timescales. Also, a single human operation sometimes translates to many computer operations (that is, round-trips). For example, opening a single Web page usually requires many round-trips, even if you are getting only a single large object (for example, a large picture).

Propagation, Bandwidth, Latencies, and Hops

The traversal of the fiber loop between New York and San Francisco presumes a data-transfer unit of a single

encoded binary digit of information. The lower bound for that traversal would be 2×19 , or 38ms (or 368 million instructions). The time for this bit to travel from its source to its destination and back again is called its propagation delay.

Propagation delay is important, but compared with the much more common metric of bandwidth—measured in bits per second—it is rarely quoted as a figure of merit. At least partially, this is because the observed propagation delay depends on context, whereas

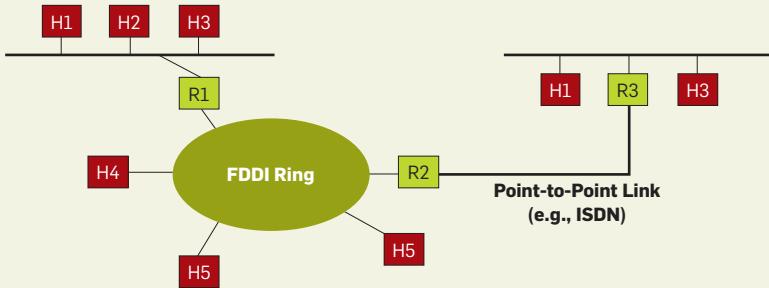
bandwidth (say of a fiber-optic transmission system) can be measured in isolation. Bandwidth can also be increased through engineering (for example, through encoding schemes for transmission systems that encode multiple bits per symbol) and thus is more attractive as a figure of merit to those who build transmission systems. Finally, bandwidth is a measure of work, which is attractive to purchasers.

Bandwidth can also affect latency, which is distinct, in my view, from propagation delay; the propagation

Table 1: Traceroute results from Klondike.cis.upenn.edu to cs.stanford.edu.

Hop	Time 1 (ms)	Time 2 (ms)	Time 3 (ms)	State
1	0.284	0.197	0.189	PA
2	0.985	0.870	0.725	PA
3	0.279	0.257	0.292	PA
4	5.065	4.856	0.544	PA
5	0.795	0.752	0.753	PA
6	2.736	2.799	2.703	PA?
7	8.329	7.810	7.795	DC
8	21.681	21.360	21.350	GA
9	44.804	44.882	44.886	TX
10	81.997	80.295	80.260	CA
11	77.328	79.228	*	CA
12	90.434	86.616	*	CA
13	86.419	86.453	*	CA
14	87.524	87.481	87.481	CA
15	87.955	87.787	87.941	CA
16	*	*	*	CA?
17	88.352	87.947	87.981	CA

Adapted from *Computer Networks: A Systems Approach* by Larry L. Peterson and Bruce S. Davie, Morgan Kauffmann, 1996.

Figure 1: Disparate network types are overcome by Internetworking technology.

delay is a metric for the first bit, while latency is a metric for the entire data unit, which may contain more than one bit. In general:

$$\text{latency} = \text{propagation delay} + \frac{\text{data unit size}}{\text{bandwidth}}$$

What this says is the propagation delay is only part of the picture and that bandwidth affects performance as well. A look at the impact of the bandwidth in an example system shows why propagation delay is so important. Consider a 10Gbps transmission system and a 1,250-byte (or equivalently, 10Kbit, chosen both to reflect a reasonable maximum transmission unit with Ethernets and to make arithmetic easier!) data unit. The propagation time for the first bit in the NY-SF loop is 38ms,

and the last bit arrives a microsecond (10K/10G) later, making the total latency 38.001ms.

The majority of the latency is propagation delay. An interesting arithmetic exercise is to compute the distance at which a transmission system's latency is double the propagation delay. For a 10Gbps transmission system and 10Kbit data unit size, this is about 214 meters, or a few city blocks. For smaller data units or longer distances, propagation delay is the majority of the latency. (More detail on propagation delay versus latency can be found in Shaffer.⁴)

It is instructive to take a few measurements to see what is what. Using the ping utility to send ICMP ECHO packets, I measured the round-trip latency between the University of Penn-

sylvania (klondike.cis.upenn.edu) and Stanford University (cs.stanford.edu)—two well-connected sites—as being about 87.5ms. Rounding this to 88ms and subtracting the fiber propagation time of 38ms leaves a difference of 50ms. (Note that the NY-SF numbers are assumed to be roughly equivalent to those from Philadelphia to Palo Alto.) Since these data units are only about 500 bits long, bandwidth between Penn and Stanford would have to be pretty bad (500 bits in 50ms would be about 10Kbps) to be the explanation. So what could it be?

There are at least two possible factors, both of which can be explained with the notion of *hops*. To understand hops, it helps to understand how a network differs from our 8,250km loop of fiber. A real network is constructed of many interconnected pieces—for example, local area networks and wide area networks. Figure 1 represents a real physical network topology, with many types of networks and multiple devices. Hosts are labeled with H, routers with R, and network types are shown to be multiple in nature.

Many different packet formats and data units are in use, and the genius of the Internet is that it has a solution to make them all work together. This *interoperability layer* consists of a packet format and an address that is interpreted by IP routers. The subnets interconnecting the routers can use whatever technology they choose as long as they can carry encapsulated IP packets between routers. Each router-router path is called a *hop*. As before with ping, it is instructive to obtain a measurement, which I did using traceroute between the two hosts I had used previously. Traceroute repeatedly sends out datagrams with a limited maximum hop count to stimulate a failure indication that can be used to determine the router. Iterating through 1 hop, 2 hops, 3 hops, and so on, gives at least an indication of the route taken by the datagrams. Inaccuracies can occur for many reasons including route changes and noncompliant router software, but it usually provides a good approximation. Table 1 illustrates data obtained from such a measurement. Using the router names output by traceroute in my sample measurement, I attempted to infer the

location of each hop, for example that atla.net.internet2.edu was in Georgia, that hous.net.internet2.edu was in Texas, and losa.net.internet2.edu was in California.

There are 17 hops reported. Our analysis of an unobstructed fiber did not account for these routers, nor for the possibility that packets did not travel “as the crow flies” between the source and destination. The total propagation delay through this network, then, is equal to the sum of the propagation time across each subnet, plus the time required to pass through the routers. This time includes both the time to switch from an input subnet to an output subnet and the additional time spent waiting in queues of packets held in memory associated with line cards. If these queues are filled with packets because an output subnet is too busy, congestion occurs, and packets that cannot be buffered are dropped.

Modern routers such as the Cisco CRS-1 exhibit average latencies of about 100 microseconds¹ when there is no queuing. Our Philadelphia–Palo Alto example would include approximately 30 of them in the round-trip path, making the total switching time latency about 3ms. The other causes of delay are more difficult to measure, but we can see that hop 8 (Atlanta) to hop 9 (Houston) takes about 23.5ms and is about 1129km. To estimate the speed, we calculate $1129000 / .0235$, which is 48042553m/sec or about 16% of the speed of light. Hop 9 (Houston) to hop 10 (Los Angeles) takes about 35.5ms to travel 2211km, which works out to a little less than 21% of the speed of light. So each hop is slowing things down quite a bit. An additional factor is routing, and the possibility of poor route selection. Routers attempt to optimize a path between two points, but that may be difficult, so in addition to the delay through the routers we can expect a certain delay caused by path selections that deviate from a straight line. An example of this can be found in Table 2, where hop 9 is in New York, hop 10 is in Massachusetts (Boston) and hop 11 (the one that takes 19ms) is in Rhode Island (Providence). Table 2 is interesting also in that it shows that about 15ms is lost in the NJ/NY area and 10ms in the California area, both

areas where not much actual distance is traveled. Other possible sources of delay include slower routers (the CRS-1 is a very high performance router) and other intervening appliances (such as firewalls) and slow links. Nonetheless, it is impressive that the IP routing infrastructure is only about a factor of two “slower” than the speed of light in fiber: 88ms vs. 38ms.

This observation of the difference between pencil and paper and measured results leads to the definition of the *throughput* of a system, which is how many bits per second you can send after taking all the real-world limitations—propagation delays, bandwidth, latency, and hops—into account.

Interprocess Communication and Protocols

In a distributed system, processes that need to communicate do so via one or more schemes for IPC.³ Example schemes include messages, reliable streams, and remote procedure calls. It is easiest to think of IPC in terms of messages, sometimes called application data units (ADUs), as they are the building blocks on which other IPC mechanisms, including reliable bytestreams, are built. Messages may require multiple IP packets. The socket API is one example of a way in which message and reliable bytestream ser-

vices can be accessed. It resembles input/output, supporting a read/write style of interface. The impact of the IPC software on a single message’s latency is typically low; ping measurements of a local loopback interface on klondike.cis.upenn.edu show times of about 20 microseconds of latency. The largest cause of propagation delays in IPC is protocols.

Protocols are rules for communicating intended to provide desired properties, such as high application throughput, reliability, or ordering. Reliable message delivery is a common application requirement and usually requires confirmation from the receiver to the sender, thus implying a round-trip. Communications requiring more data than a single packet must use multiple packets, implying multiple round-trip times. To see the impact of the physics on a naïve protocol, imagine an IPC system that uses 10Kbit packets and must move 100Kbits (10-packets worth of data) across the U.S., which as we have seen (for a single transcontinental piece of fiber) should require about 19ms. If a new packet is sent only when a previous one has been acknowledged, one packet will be sent every 38ms, and the communication will require 380ms, or almost one half second, independent of the bandwidth of the network. Yet, it’s clear that with

Table 2: Traceroute results from home network (in N.J.) to cs.stanford.edu.

Hop	Time 1 (ms)	Time 2 (ms)	Time 3 (ms)	State
1	1.835	0.594	0.461	NJ
2	5.336	3.634	5.076	NJ
3	5.043	4.093	4.975	NJ
4	10.043	9.408	9.656	NJ
5	14.687	14.319	14.902	NY
6	15.025	14.309	18.432	NY
7	14.004	14.254	15.001	NY
8	14.914	13.945	14.916	NY
9	14.985	14.031	16.568	NY
10	19.003	18.738	18.890	MA
11	19.986	39.004	41.033	RI
12	59.984	58.776	58.969	?
13	41.168	90.035	88.657	IL?
14	60.013	89.862	88.904	CA?
15	89.992	89.749	91.322	CA
16	90.302	97.151	96.555	CA
17	*	92.022	*	CA
18	97.272	98.329	99.830	CA

a high-throughput network, one could have sent all 10 of the packets in a row and waited for a confirmation that all 10 arrived, and this could be done in 38ms.

This example along with Figure 2 illustrates what is often called the *bandwidth-delay product*, which is a measure of the capacity of a path in bits between a source and a destination. Figure 2 shows there may be usable capacity not being used, illustrated here by the spaces between packets. If the network were fully utilized, then all of the capacity would be fully occupied by packets in flight. When the network is fully occupied with packets, a bandwidth-delay product of bits will be in flight between a source and destination. The challenge is estimating the available capacity at any given time, as network dynamics could make this estimate highly variable. If we overestimate the capacity, too many packets will be pushed into the network, resulting in congestion. If we underestimate the capacity, too few packets will be in flight and performance will suffer.

Optimizing protocols to the available bandwidth-delay product has been a long-standing problem of interest to the networking community, resulting in many algorithms for flow control and congestion control. TCP/IP, for example, uses acknowledgments from the receiver to pace the sender, opening and closing a window of unacknowledged packets that is a measure of the bandwidth-delay product. If a packet loss occurs, TCP/IP assumes it is congestion and closes the window. Otherwise, it continues trying to open the window to discover new bandwidth as it becomes available.

Figure 3 shows how TCP/IP attempts to discover the correct window size for a path through the network. The line indicates what is available, and this changes significantly with time, as competing connections come and go, and capacities change with route changes. When new capacity becomes available, the protocol tries to discover it by pushing more packets into the network until losses indicate that too much capacity is used; in that case the protocol quickly reduces the window size to protect the network from overuse. Over time, the “sawtooth” re-

Many different packet formats and data units are in use, and the genius of the Internet is that it has a solution to make them all work together. This interoperability layer consists of a packet format and an address that is interpreted by IP routers.

flected in this figure results as the algorithm attempts to learn the network capacity.

A major physics challenge for TCP/IP is that it is learning on a round-trip timescale and is thus affected by distance. Some new approaches based on periodic router estimates of available capacity are not subject to round-trip time variation and may be better in achieving high throughputs with high bandwidth-delay paths.

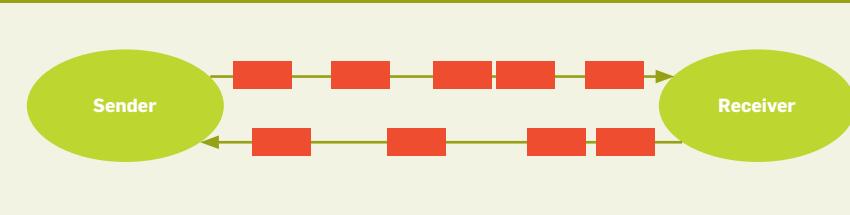
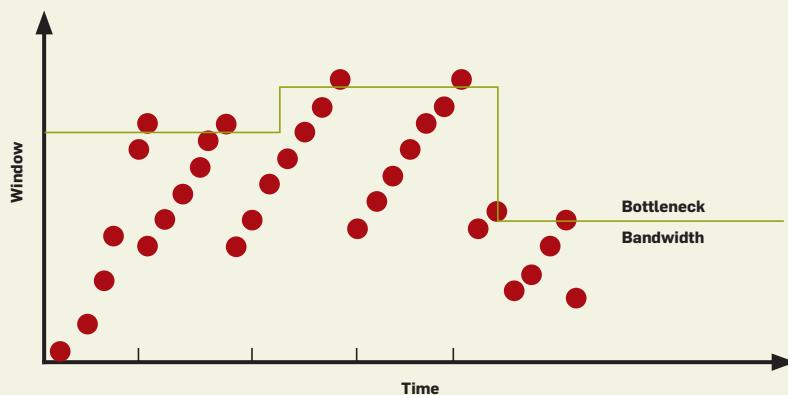
Implications for Distributed Systems

Many modern distributed systems are built as if all network locations are roughly equivalent. As we have seen, even if there is connectivity, delay can affect some applications and protocols more than others. In a request/response type of IPC, such as a remote procedure call, remote copies of data can greatly delay application execution, since the procedure call is blocked waiting on the response. Early Web applications were slow because the original HTTP opened a new TCP/IP connection for each fetched object, meaning that the new connection's estimate of the bandwidth-delay was almost always an underestimate. Newer HTTPs exhibit persistent learning of bandwidth-delay estimates and perform much better.

The implication for distributed systems is that one size does not fit all. For example, use of a centralized data store will create large numbers of hosts that cannot possibly perform well if they are distant from the data store. In some cases, where replicas of data or services are viable, data can be cached and made local to applications. This, for example, is the logical role of a Web-caching system. In other cases, however, such as stock exchanges, the data is live and latency characteristics in such circumstances have significant financial implications, so caching is not effective for applications such as computerized trading. While in principle, distributed systems might be built that take this latency into account, in practice, it has proven easier to move the processing close to the market.

Rules of Thumb to Hold Your Own with Physics

Here are a few suggestions that may

Figure 2: Packets in flight between a sender and a receiver.**Figure 3: TCP/IP attempts to discover the available network capacity.**

help software developers adapt to the laws of physics.

Bandwidth helps latency, but not propagation delay. If a distributed application can move fewer, larger messages, this can help the application as the total cost in delay is reduced since fewer round-trip delays are introduced. The effects of bandwidth are quickly lost for large distances and small data objects. Noise can also be a big issue for increasingly more common wireless links, where shorter packets suffer a lower per-packet risk of bit errors. The lesson for the application software designer is to think carefully about a design's assumptions about latency. Assume large latencies, make it work under those circumstances, and take advantage of lower latencies when they are available. For example, use a Web-embedded caching scheme to ensure the application is responsive when latencies are long, but no cache when it's not necessary.

Spend available resources (such as throughput and storage capacity) to save precious ones, such as response time. This may be the most important of these rules. An example is the use of caches, including preemptive caching of data. In principle, caches can be replicated locally to applications, causing

some cost in storage and throughput (to maintain the cache) to be incurred. In practice, this is almost always a good bet when replicas can be made, because growth in storage capacities and network throughputs appears to be increasing at a steady exponential rate. Prefilling the cache with data likely to be used means that some capacity will be wasted (what is fetched but not needed) but that the effects of some delays will be mitigated when predictions of what is needed are good.

Think relentlessly about the architecture of the distributed application. One key observation is that a distributed system can be distributed based on function. To return to the design of a system with a live data store (such as a stock market), we might place the program trading of stocks near the relevant exchanges, while placing the user interaction functionality, account management, compliance logging, etc. remotely in less exchange-local real estate. Part of such a functional decomposition exercise is identifying where latency makes a difference and where the delay must be addressed directly rather than via caching techniques.

Where possible adapt to varying latencies. The example of protocols maximizing throughput by adapting to

bandwidth-delay capacities shows how a wide range of latencies can be accommodated. For distributed applications, this might be accomplished by dynamically relocating elements of a system (for example, via process migration or remote evaluation).

None of these suggestions will allow you to overcome physics, although prefetching in the best of circumstances might provide this illusion. With careful design, however, responsive distributed applications can be architected and implemented to operate over long distances.

Summary

Propagation delay is an important physical limit. This measure is often given short shrift in system design as application architectures evolve, but may have more performance impact on real distributed applications than bandwidth, the most commonly used figure of merit for networks. Modern distributed applications require adherence to some rules of thumb to maintain their responsiveness over a wide range of propagation delays. □

References

1. *Light Reading*, 40-gig router test results; http://www.lightreading.com/document.asp?doc_id=63606&page_number=4&image_number=9.
2. Mohr, P.J., and Taylor, B.N. CODATA recommended values of the fundamental physical constants. *Reviews of Modern Physics* 77, 1 (2005), 1–107.
3. Partridge, C. *Gigabit Networking*. Addison-Wesley Professional, 1994.
4. Shaffer, J.H., and Smith, J.M. A new look at bandwidth latency tradeoffs. University of Pennsylvania, CIS TR MS-CIS-96-10; http://repository.upenn.edu/cgi/viewcontent.cgi?article=1192&context=cis_reports.

Related articles on queue.acm.org

You Don't Know Jack about Network Performance

Kevin Fall and Steve McCanne
<http://queue.acm.org/detail.cfm?id=1066069>

Latency and Livelocks

Kode Vicious
<http://queue.acm.org/detail.cfm?id=1365494>

DNS Complexity

Paul Vixie
<http://queue.acm.org/detail.cfm?id=1242499>

Jonathan M. Smith is the Olga and Alberico Pompa Professor of Engineering and Applied Science and a professor of computer and information science at the University of Pennsylvania. He served as a program manager at DARPA from 2004 to 2006 and was awarded the Office of the Secretary of Defense Medal for Exceptional Public Service in 2006.

contributed articles

DOI:10.1145/1538788.1538809

Network software adapts to user needs and load variations and failures to provide reliable communications in largely unknown networks.

BY EROL GELENBE

Steps Toward Self-Aware Networks

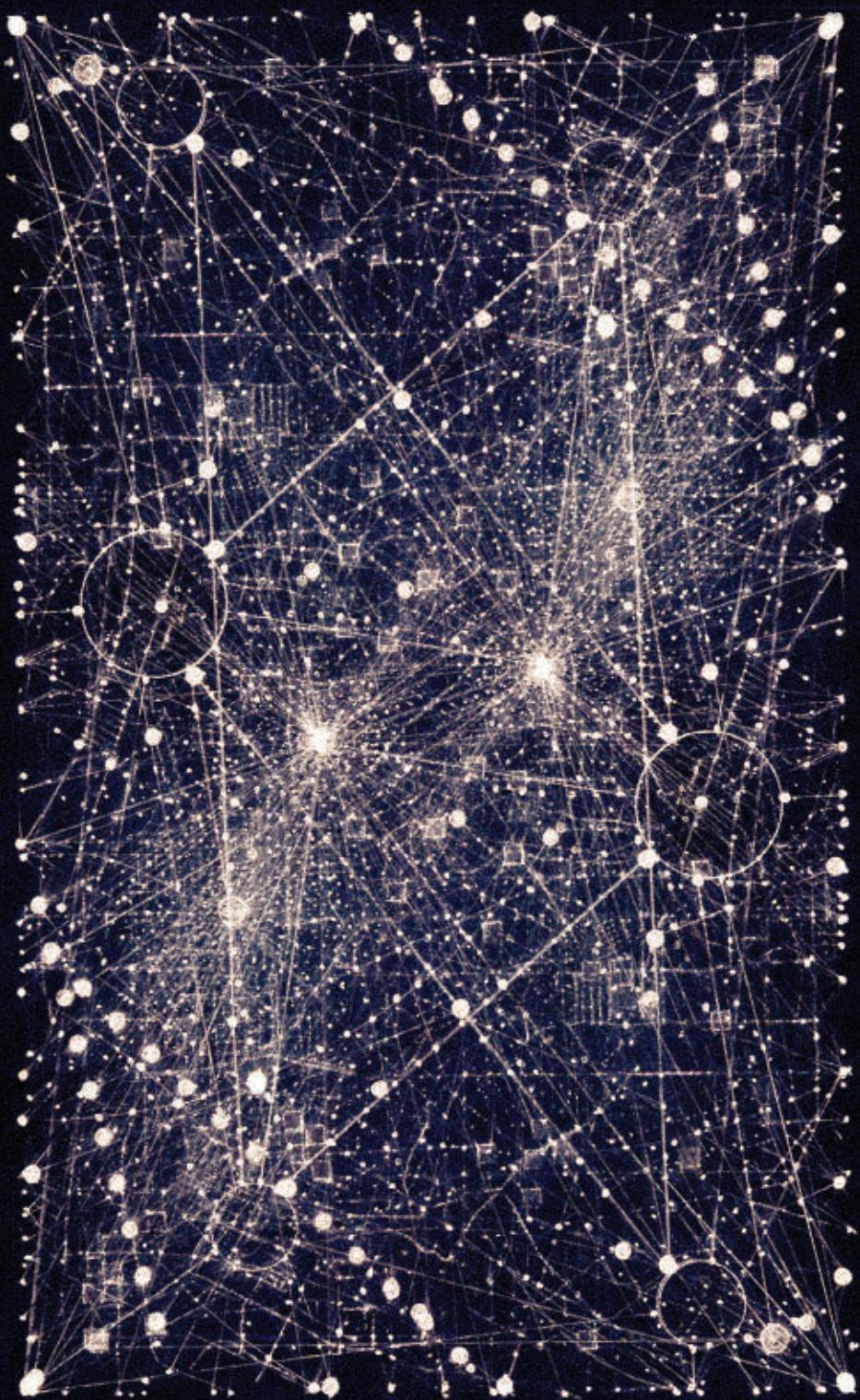
THE INFORMATION NEEDED to route packets in large networks and in networks in which nodes join and leave the network frequently or move in and out of wireless range of each other can change more frequently than the rate routing information is updated throughout the network. In such a system it becomes necessary to allow individual nodes to proactively discover the presence of other nodes, links, and paths (as needed and on demand), leading to the design of self-aware networks. Here, I focus on experimental and theoretical research concerning the technical steps leading to these networks.

The Internet Protocol offers an orderly update of its status based on the shortest-path algorithm,¹⁸ Distance Vector,²² and Link State²³ techniques so routing algorithms operate seamlessly, despite changes in network topology and conditions. However, as computer networks become extremely large, the information available concerning the network state becomes uncertain. Link state changes

are more frequent in larger networks, increasing the overhead and delay due to updates throughout the network. Consequently, information about the network state, including connectivity, condition of nodes, traffic conditions, and quality of service (QoS), propagates more slowly than rate changes occur. The need to convey time-sensitive information (such as voice and media) also motivates investigation of routing techniques based on user requirements and the network's instantaneous state. Thus it is preferable that nodes discover the network state autonomously, without having to rely on an overall scheme that updates routing tables systematically throughout the network. Information updates can be initiated by the nodes that need this information at the time it is needed, rather than throughout the network and when changes occur.

We use the term "self-aware network," or SAN,¹¹ for a system consisting of nodes that can join and leave the network autonomously and discover paths when the need to communicate arises. The nodes in a SAN should sense the status of other nodes, links, and paths, including traffic level and congestion, so as to update their own relevant information about the paths they need to use, based on criteria specific to their own needs. Each connection may then use paths that optimize the connection's own QoS criteria, rather than a common criterion (such as the shortest path) for all connections. These needs might include user QoS requirements, or performance, reliability, security, defense against attacks,^{9,24} and power utilization.¹² A SAN can be a wired, wireless, or a peer-to-peer system. A wireless ad hoc network is a practical example of a SAN that responds to time-varying conditions related to the mobility of nodes and changes in the conditions of wireless links (such as noise and physical obstructions). Networks that must operate autonomously and remotely (such as sensor networks) also benefit from self-aware capabilities.

Research on effective SAN architec-



tures also motivates work on autonomic communications⁵ and bio-inspired techniques for networking. Ideally, self-awareness is a desirable property of most networked systems. However, for SANs to be widely accepted, many fundamental questions must be answered affirmatively, including:

a. Assuming that in the worst case a node knows only its immediate neighbors (though the network is fully connected), can a node forward a packet successfully to any other node in the SAN in finite time without routing tables at each node?

b. What are practical means for gathering information about communication paths without flooding the network with requests for information

and with replies to these requests? Is it possible to constantly improve the accuracy of the information being gathered (in the presence of time-varying network conditions) in a way that focuses on the information that is actually needed, rather than trying to gather information about all possible paths?

c. Can self-awareness be exploited for timely decision making without risking the consequences of constant “changes of mind”? For instance, distinct nodes could select the same path in an uncoordinated manner due to the fact it is momentary, then have to renege when all use it and hence overload it. What are the risks, costs, and mitigating factors associated with frequent “oscillations” regarding such decisions?

Other relevant questions involve scaling, security, reliability, and mobility. Our work at Imperial College has shown that security^{9,24} and reliability (discussed later) can also be enhanced in a SAN. However, the effect of malicious nodes and users and node mobility (often studied in mobile ad hoc networks¹²) need further work. Scalability of SANs can be improved through recursive routing²¹ and hierarchical routing techniques that have long been used in the Internet.

Reliable Communication in Unreliable Networks

Travel time in unknown environments is of interest in networks and robotics; algorithms that minimize worst-case travel times in finite graphs were covered by Papadimitriou and Yannakakis.²⁵ The first question (a) raised earlier is answered by our result showing that average travel time is finite under worst-case conditions in an infinite graph,⁸ as long as packet forwarding can be aborted when the packet is unable to reach the destination after a predetermined length of time, and the forwarding process is then restarted at the source, provided that the routing process is randomized. This proves that packets can be reliably forwarded to destinations with probability one, even when routing information is not available, provided that a randomized algorithm is used.

Consider some node U that wishes to forward a packet to a destination V to which there exists at least one valid path. However, we admit the possibility of errors in the routing information about how to reach V , allowing for approximate or erroneous routing. Since the network is infinite and nodes may not know the direction a packet needs to be forwarded, a packet can get lost and meander indefinitely from node to node without ever reaching its destination. Let us make things worse by also allowing packets to be dropped inadvertently. The system uses a time-out, so if a packet does not reach its destination before the time-out elapses, the packet is destroyed and retransmitted by the source. Since there is at least one path from U to V , the shortest path length D (expressed in number of hops) is finite. The mathematical model for such a system is a random walk, where the

Figure 1: A testbed.

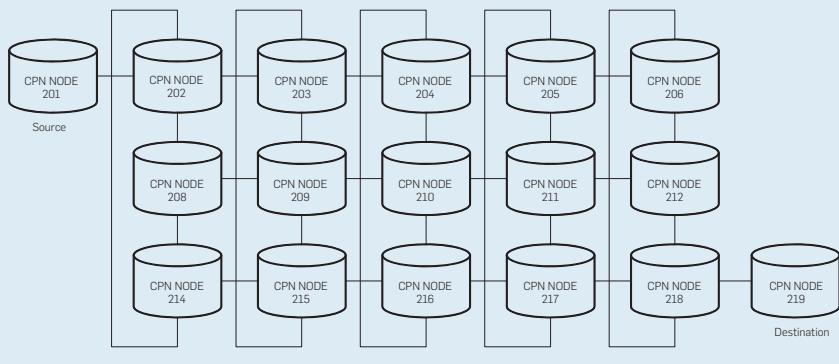
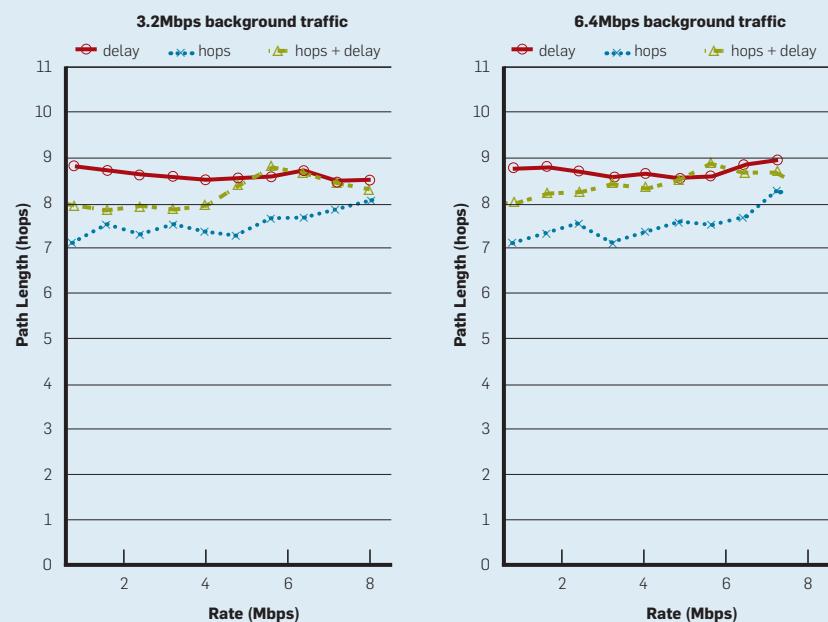


Figure 2: Path length compared.



“walker” is a packet being forwarded from U to V , starting at U at time $t = 0$. The packet’s remaining distance X_t at time t is the length of the shortest path to the destination, and the travel time is the first instant T when $X_T = 0$. The key question—whether there exists a finite T such that $X_T = 0$ —is answered by Gelenbe⁸ showing that:

$$E[T] = 2D \frac{1+r}{-b + \sqrt{b^2 + 2c(\lambda+r)}}$$

Here b is the “drift” parameter, so if $b < 0$ then the packet is on the average making progress toward the destination, while if $b > 0$ then it is moving away from it, and c is the variance per unit time or fluctuation related to the packet’s motion toward or away from the destination, so $c > 0$. $1/r$ is the average value of the time-out, and λ is the probability per unit time that the packet is lost. The expression above tells us, as expected, that if there is no time-out, that is, $r = 0$, and losses are possible, that is, $\lambda > 0$, then $E[T] = +\infty$, that is, a packet will never make it to its destination. If there are no packet losses, that is, $\lambda = 0$, and there is also no time-out, that is, $r = 0$, then $E[T] < \infty$ if $b < 0$, while if $b > 0$ then, as expected, $E[T] = +\infty$. The time-out is also thus a protection against packets that “lose their way” by traveling on and on through the infinite network without ever reaching their destination. Most interestingly, when $c > 0$, that is, there is randomness in the path, we have $E[T] < \infty$ as long as there are losses or a finite time-out. However, if the path is deterministic, that is, $c = 0$, then the travel time is infinite unless $b < 0$. Thus we establish that, even in the worst case of an infinitely large network in which individual nodes may lose packets and packets may lose their way by meandering indefinitely in the network, as long as there is randomness in the routing ($c > 0$) and a finite time-out is available ($r > 0$), the packet will reach its destination in finite time, even though no correct routing information is available at the nodes of the network. This model also covers the case of “wrong” routing information with $b > 0$, where packets are probabilistically sent away from the destination, and with uncertain or “partially correct” routing information with $b < 0$ where (on average)

Interesting is that the criterion that combines delay with number of hops leads to the best results, though they are comparable to the results based on using just the delay as the QoS goal.

packets get closer to the destination at each step. The “ideal” case $b = -1$ is when the packet makes the fastest possible progress to the destination.

Self-Aware Routing

The second question (b) concerns the routing algorithms. Most routing techniques attempt to optimize one or more criteria in addition to the basic requirement of forwarding traffic from any source to any destination. The shortest-path routing algorithm is based on the premise that if a packet visits the smallest possible number of hops toward its destination, then the network overhead is minimized, as is most of the other criteria of interest (such as packet loss and packet delay). A SAN will attempt to optimize network performance through exploration, measurement, and adaptation, rather than through an a priori choice (such as the shortest path).

Much of the published work on SAN routing follows two approaches using reinforcement learning (RL), first proposed for packet routing by Boyan and Littman.² The Cognitive Packet Network (CPN) approach^{11,13} uses “smart packets” (SP) for path discovery, together with RL and neural networks installed in each network node, adaptively selecting paths so as to offer “best effort” QoS to end users. SPs are sent out by nodes that are actively involved in forwarding packets to discover and assess paths that lead to destination nodes. The “Ant Colony”^{3,4,19} paradigm searches for paths from source nodes to specific destination nodes by emulating the pheromone-based technique used by biological ants to mark their paths and communicate with fellow members of the same colony. Both CPN and Ant Colony algorithms include random search when information about suitable paths is unavailable, reinforcing the importance attributed to paths that appear to be best and using alternate paths when previously selected paths prove less desirable.

In CPN, SPs discover routes for connections to specific destinations. They are routed using RL based on a QoS “goal.” We use the term “goal” to indicate that there is no guaranteed QoS and that CPN provides a best effort to satisfy the desired QoS. SPs do not carry payload, finding routes and col-

Figure 3: Total packet delay with 6.4Mbps background traffic carried out for SPs, that is, for a small fraction of the traffic, resulting in reduced router computation overhead.

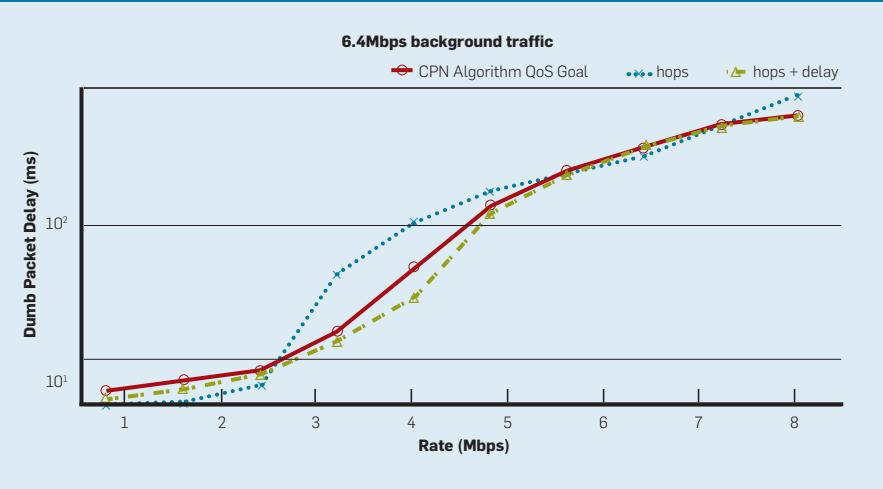
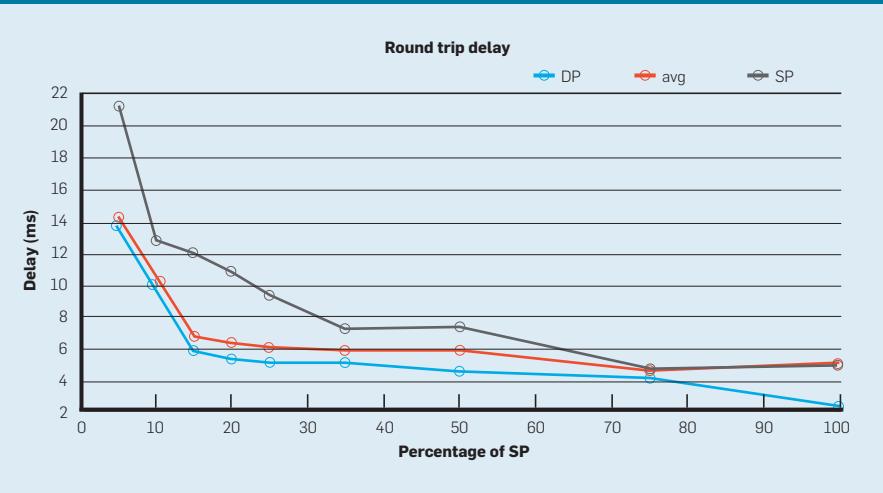


Figure 4: Total average delay for SPs (top) and DPs (bottom) and average delay for all packets (center) as a function of the percentage of SPs.



lecting measurements based on three complementary elements:

- Each node engaged in forwarding packets to some destination sends out SPs that search for paths to the destination(s) and gather measurement data about these paths. This data is not limited to delay and packet loss but may also include measurable information about power utilization by nodes on those paths, the volume of traffic on the paths, and the security of the nodes and links on the paths. SPs do not carry the actual traffic payload but are used just for measurement and exploration;

- Each node maintains a neural network to compute the next node an SP from this node must go to. The weights of the neural network are updated using an RL algorithm that uses data collected by the SPs. In CPN, the role of the

neural network is just to route the SPs, and the “dumb packets” (DP) that carry the payload are routed differently; and

- Each source node maintains an ordered list of paths to the destination(s) they are concerned with. This list includes paths that are discovered by the SPs and is updated using the QoS information collected by the SPs. The list is ordered with the best paths at the top, so the payload or DB is forwarded along the complete path (that is, they are source-routed), and intermediate nodes do not normally interfere with them other than providing a store-and-forward capability.

When (and if) an SP arrives at its destination, the destination generates an acknowledgment (ACK) packet, and the ACK stores the “reverse route,” as well as the measurement data collected by the SP. An SP that does not reach

its destination after a predetermined number of hops (typically set as a multiple of the network’s diameter, here 30) is destroyed. The ACK being returned as a result of an SP will travel along the “reverse route” obtained from the SP’s route, examining it from right (destination) to left (source), removing any sequences of nodes that begin and end in the same node. For instance, the path $\langle a, b, c, d, a, f, g, h, c, l, m \rangle$ will result in the reverse route $\langle m, l, c, b, a \rangle$. Note that the reverse route is not necessarily the shortest reverse path nor the one resulting in the best QoS. Also ACKs deposit QoS measurement data in mailboxes (MBs) at the nodes they visit as they move toward the SP’s source node. On the other hand, DPs carry payload and use dynamic source routing. The route brought back by an ACK is used as a source route by subsequent DPs of the same QoS class with the same destination until a new route is brought back by another ACK. An MB in each node stores QoS information.

Each MB is organized as a least-recently-used (LRU) stack. The entries in an MB are identified with the QoS class and the destination. Since SPs are routed at each node using RL, they concentrate their search on the most promising paths for a given destination. Each node contains one or more random neural networks (RNNs),¹⁵ where each RNN corresponds to a QoS class and a destination. In the RNN, the choice of the output link of a node is represented by a neuron, and the link corresponding to the “most excited” neuron is used to forward a given SP. The RL algorithm operates as follows:

A “goal function” G is used to characterize the objective one wishes to optimize for a given source to destination connection; this objective may be hop count (if one wants to minimize path length), delay, packet loss rate, energy utilization, and more, or a combination of these factors. The reward R , which is defined as $R = 1 \div G$, and successive values of R obtained from measurements carried back by the ACK packets, are denoted by R_l , $l = 1, 2, \dots$, and used to compute a “historical value” of R :

$$T_l = \alpha T_{l-1} + (1 - \alpha) R_l$$

where α is some constant ($0 < \alpha < 1$) that determines the algorithm’s mem-

ory, and R_l is the most recently measured value of the reward. Suppose we have made the l th decision that chooses the output link (neuron) j , where the l th reward calculated for the QoS information received from the network is R_l . We first determine whether R_l is larger than or equal to the threshold T_{l-1} . If it is, then to reward this success, we increase (significantly) the excitatory weights going into neuron j and make a small increase in the inhibitory weights leading to other neurons. If the R_l is less than T_{l-1} , then we moderately increase the excitatory weights leading to all neurons other than j to open up different decision options and increase significantly the inhibitory weight leading to neuron j in order to punish it for not having provided a useful prediction.

Finally, the excitation probabilities of each neuron in the RNN are computed, and the SP is forwarded to the output link corresponding to the neuron that is the most “excited.” The arrival of an ACK to a node triggers the update of the weights of the RNN, while the arrival of an SP to the node triggers the execution of the RNN algorithm to make the routing decision. Thus several weight updates can occur between two successive updates of a routing decision. Similarly, if no ACKs arrive at a given node between two successive arrivals of an SP, the successive SPs will use the same routing decision.

Numerous experiments have been run with CPN with both simulation and actual network testbeds;^{11,13,14} here, we report on three with real networks with 17, 25, and 46 nodes and different topologies. All measurements we report used testbeds built with off-the-shelf components running CPN. The routers were Pentium IV-class machines with four-port Ethernet interfaces running Linux 2.6.15, where CPN was implemented as a loadable kernel module. All links were full-duplex at 10MB/sec or 100MB/sec, depending on the experiment.

We start with measurements made in a wired testbed consisting of 17 nodes (see Figure 1) chosen because it offers a large number of alternate paths within a relatively small network; adjacent nodes are connected with 10Mbps Ethernet links. All tests use a flow of UDP packets entering the CPN

network with constant bit rate (CBR) traffic and packet size of 1,024KB. For each experiment, 10,000 packets were sent out from source to the destination, and each measurement point provided averages or statistics for the 10,000 packets when background traffic is in-

troduced to each link. The average hop count, forward delay, and packets loss rate under different background traffic were reported. We used Algorithm-H, Algorithm-D, and Algorithm-HD to denote the RNN routing algorithms using hop, delay, and the combination

Figure 5: Use of routes with low traffic rate; delay is the QoS goal.

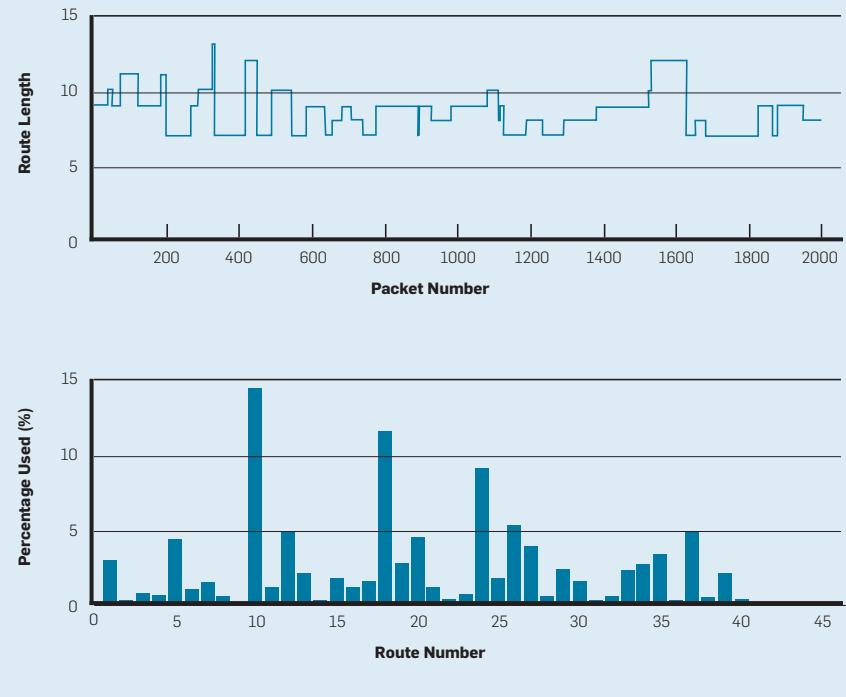
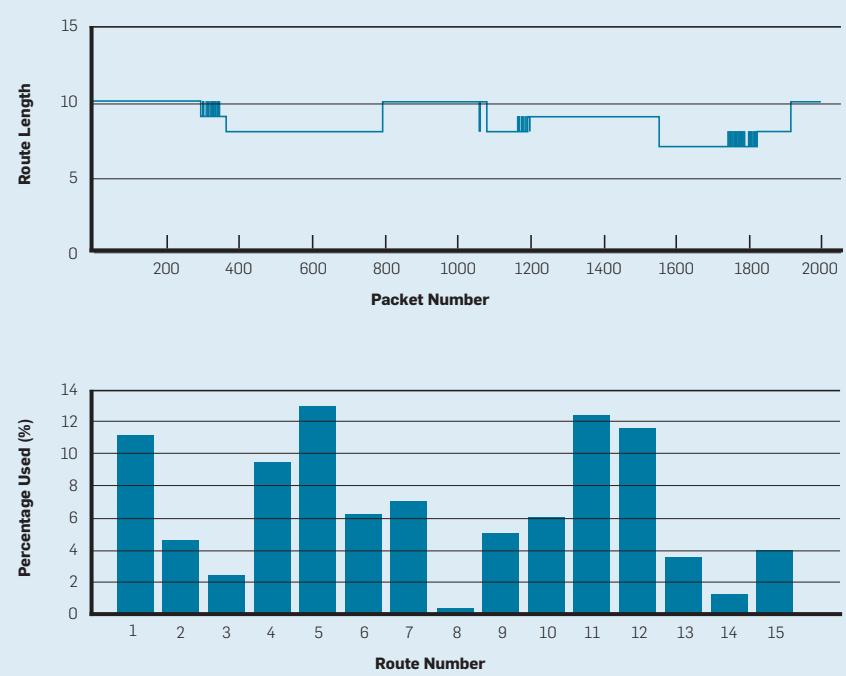


Figure 6: Use of routes with high traffic rate; delay is the QoS goal.



of hop count and delay as the QoS goal, respectively. The length of the shortest path between the source (201) to the destination (219) was seven hops; there were five different shortest paths.

Figure 2 shows that CPN indeed provides the “self-aware” capability being sought since its measured behavior corresponds to the stated QoS goal. The curves give the average number of hops of the routes CPN selects when different QoS goals are used with different levels of background traffic (distinct figures), while end-to-end traffic is varied along the x -axis. When hop count is

used as the QoS goal (cross or “hop” in the figure), the average number of hops under different background traffic rates are close to the minimum value of seven hops. When delay is used as the QoS goal (circle or “delay” in the figure), the average path length is longer, so CPN adapts to the guidelines it has received and chooses shortest-delay paths rather than shorter hop paths. Note that when the source-to-destination traffic is high (right-hand side of the figure) there is little difference in path lengths for the different QoS goals, due to the fact that performance

is equally poor for all possible criteria in heavy traffic.

The average packet-forwarding delay for each of the goal functions (see Figure 3) is also measured as a function of the amount of traffic from source to destination for different levels of background traffic; the results confirm those in Figure 2. The blue curve in Figure 3 corresponds to using the number of hops as the QoS goal to be minimized; as expected, it leads to the longest delay. Interesting is that the criterion that combines delay with number of hops leads to the best results, though they are comparable to the results based on using just the delay as the QoS goal. Confirming the results in Figure 2, these results show that the CPN algorithm is indeed self-aware in that it is able to translate its overall objectives into effective adaptive decisions taken in real time.

Measuring total delay, including queueing and forwarding delay, experienced by SPs and DPs, one sees that (see Figure 4) when the SPs are increased (expressed as a percentage of the DPs being forwarded), the overall QoS improves, but most of the improvement is achieved at a relatively low 20% of SPs with respect to DPs. As one would hope, the DPs experience better QoS; the SPs “pay the price” of the search activity by experiencing less-favorable QoS. Since SPs and ACKs are each approximately 10% of the length of a full Ethernet packet, for 20% of SPs over DPs, the total additional traffic generated by CPN over and above the payload traffic is 0.04%. Note that route computations in CPN are carried out only for SPs, that is, for a small fraction of the traffic, resulting in reduced router-computation overhead.

To measure whether CPN spreads traffic among many paths as the traffic rate increases from 100 packets/sec in Figure 5 to 1,000 packets/sec in Figure 6, there is a more even distribution of traffic over a smaller number of paths having better QoS.

Adverse effect of slower decisions. One obvious trade-off in any decision process is whether it is better to “optimize more and decide later” or provide decisions as soon as they can be formulated and hope for the best. Thus the experiments described earlier refer to a situation where decisions were taken

Figure 7: CPN 46-node testbed subject to failures.

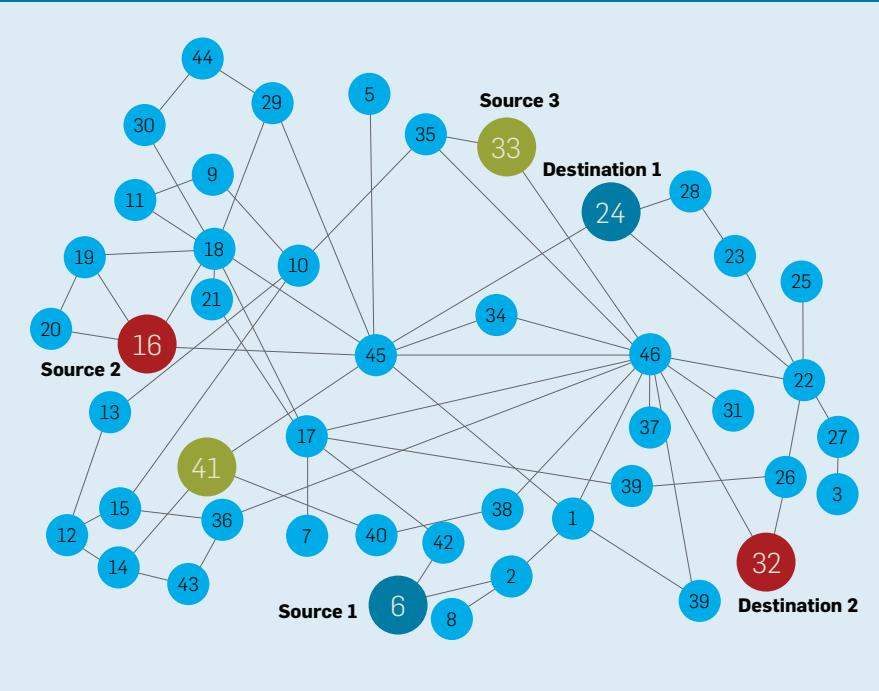
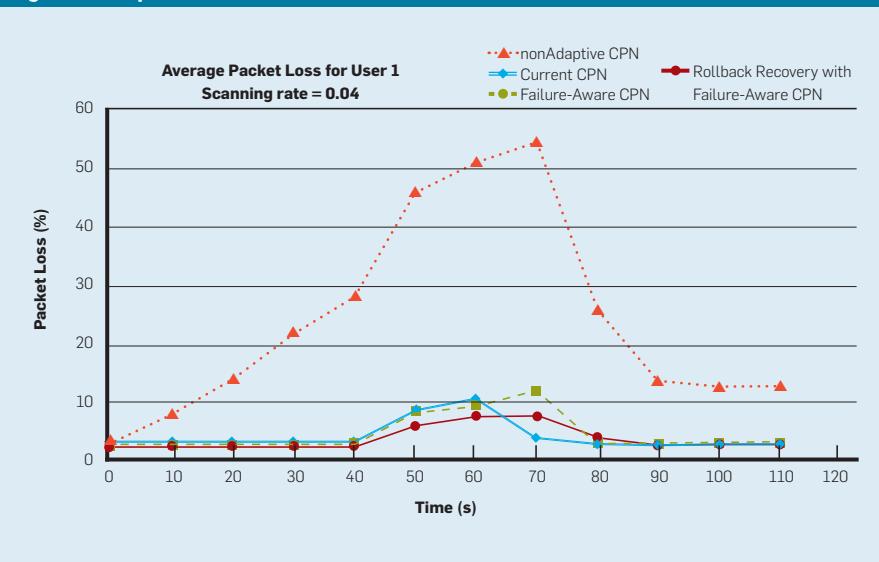


Figure 8: Adaptation reduces loss when failures occur.

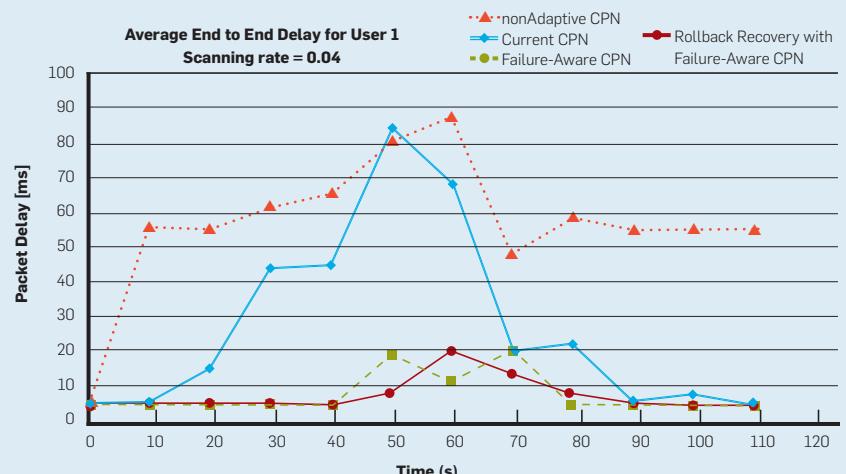


in real time based on either the current state of the RNN in each node or on the most recent RL updates that have been made. A more sophisticated way of selecting paths could use the underlying RNN and RL but also make a further optimization decision based on the fact that at each source node, CPN maintains a set of paths with the most recent measured QoS metric for each path, as well as for each of the destinations with which the source communicates. Now suppose that two distinct paths $SxIyD$ and $SuIvD$ connect source S to destination D through the same intermediate node I , and these paths have been discovered by SPs and provide QoS metrics $G(SxIyD)$ and $G(SuIvD)$. Obviously $SuIyD$ and $SxIvD$ are also valid paths. If they were not previously discovered by SPs, one can still infer their estimated (not measured) QoS assuming the QoS data is additive. We denote the inferred QoS values by $g(SuIyD)$ and $g(SxIvD)$. Now suppose that one of the two inferred QoS values, say, $g(SuIyD)$ is the “best” one (such as smallest delay, smallest loss, or best value of some other metric of interest). One can then use the hitherto untested path $SuIyD$ to forward DPs, rather than the best of the two paths actually tested. Note that this operation of selecting new paths by combining prefixes and suffixes of previously discovered paths resembles the “crossover operation” in a genetic algorithm.¹⁰ Experiments run with 1,024B DPs and varying the DP rate of 100 to 800 packets/sec showed that this additional optimization provided a small improvement in both packet loss rate and average packet delay.

However, when a significant amount of background traffic was added to each link, even with relatively low DP rate exceeding a certain value (300 packets/sec), the original CPN algorithm performed significantly better, showing that the slower optimization process using older data introduced on top of CPN by the genetic algorithm-like approach is unable to respond quickly enough to changing network conditions.

Self-aware adaptation to failures. During experiments conducted in a 46-node testbed (see Figure 7), we observed that CPN can also protect a network against worm-like failures. In these experiments, failures begin at a given node that then randomly causes

Figure 9: Adaptation reduces delay in the presence of failures.



other nodes to fail. A node that fails is unable to forward traffic, causing neighbors to fail. Node failure is followed by recovery, representing cleaning and patching, at a constant rate of 1 node/sec. Figures 8 and 9 report the measured average packet loss and delay for 10 experiments where User 1 sent 7MB/sec CBR traffic from Node 6 to Node 24. When CPN is operating, the QoS is significantly better than when CPN is stopped after paths are established (top curves). Moreover, as further adaptive measures are taken (other curves),²⁷ performance and QoS improve further.

Ant colony routing. Ant colony routing algorithms^{3,4,19} differ from CPN in the way they use RL, as well as in other respects. Inspired by the way ants use pheromones to mark their paths and communicate about sources of food, packets represent ants, nodes and links represent locations, and packets move toward their destinations based on paths with strong markings. When a packet reaches its destination, a corresponding “marking” packet heads back to the source by following the path in reverse or quasi-reverse order (or following the “strongly marked trail” and strengthening the marking at each link and node it visits). The markings degrade over time (“forgetfulness”) if not reinforced by the passage of other packets. The algorithm is initiated by a random search until the destination node is found and the discovered path(s) is reinforced by the re-

turning packets. The returning packet from the destination is like the ACK packet in CPN, but ant colony routing algorithms use payload packets for both search and data delivery, while CPN separates the search role via SPs from the payload role via DPs; the resulting QoS is better when DPs specialize in payload conveyance and SPs are restricted to search. Thus CPN uses more packets, since SPs are constantly being sent forward to accomplish the search function, representing a constant fraction (such as 10%) of total traffic. Moreover, ant colony algorithms do not use a neural network (as in CPN) to store RL information inside a given node. CPN routers carry out route computation only for SPs and represent a small fraction of total traffic, but ACKs and DPs are source-routed, while ant colony algorithms typically require route computation for all packets. Clearly, ant colony algorithms are better adapted to networks that experience frequent disruption, since all packets are in a sense autonomous. In CPN, if a DP’s path is disrupted, the packet must be retransmitted at the source, with information brought back by a subsequent SP that finds another path. Thus CPN will be better at forwarding packets but slower in responding to changes in topology.

Route Oscillations

Since the days of the ARPANET, it has been observed that route oscillations¹⁸ can cause performance to suffer under

Figure 10: CPN testbed emulating the Swiss Education and Research Network. The square node is the sink; the 24 round nodes are sources.

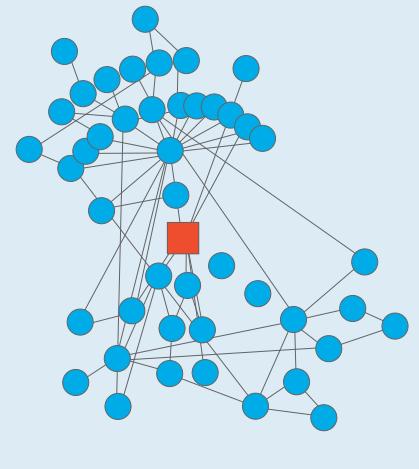
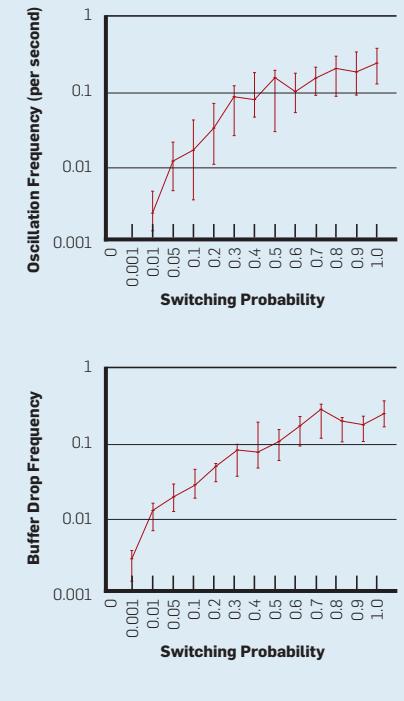


Figure 11: Oscillation frequency (top) and packet-drop rate (bottom) vs. switching probability.



medium to high load conditions. Oscillations occur if load-sensitive metrics are used to select routes, becoming more frequent at higher loads²⁸ due to the transfer of flows to lightly loaded paths that then become overloaded; the result is that flows are transferred to other paths that in turn get overloaded. The overlap of different routers' measurement windows also lead to oscillations when flows interfere and

prevent the network from stabilizing. Frequent route switching can reduce performance by slowing the network's convergence to best paths,⁶ increasing node overhead. Route oscillations also affect TCP²⁶ due to TCP response to asymmetric paths (when a data packet's path is different from that of its ACKs) and to out-of-order packet delivery when packets take different paths and reach their destinations in a different order. The output node must then reassemble packets into the right order,¹ causing additional delay and loss of packets due to the finite capacity of the buffers used for reassembly; QoS is thus degraded for real-time applications (such as voice and media). Routing oscillations are also studied in overlay networks.¹⁷

One must therefore examine whether (i) frequent oscillations can occur in a SAN, (ii) whether they can be easily mitigated or reduced, and (iii) whether they are necessarily detrimental to performance. Concerning (iii), each time a path is selected, CPN forwards the traffic along that path until the path is changed, and as long as the path is being used, useful work is done and packets are delivered to the destination. When a path switch occurs, packets already engaged in the path continue flowing to the destination along the previous path, since each DP stores the path it is following, and the change in path decided at the source affects only subsequent packets, not those already engaged in the path. Thus CPN does limit the effect of path switching in a SAN.

Concerning (ii), various ways are available to mitigate or reduce oscillations. For instance, the source node can allow switching only if the QoS gain exceeds a significant threshold. Another approach is to require that each time a path is used, that usage must exceed a certain number of packets before switching can be considered again.

Here, we report on a testbed with full-duplex links at 10Mb/sec running CPN (see Figure 10) that emulates the topology of the Swiss Education and Research Network (as of 2007)¹⁶; it included 24 constant bit-rate flows at 1.66Mb/sec generated to create DP traffic of just over 40Mb/sec. Combined with SP and DP traffic, the result was a slightly overloaded system. SP

traffic was set at 10% of DPs, and each source that sent traffic to the same destination had four inputs for a total of 40Mb/s of available incoming bandwidth. The QoS goal used was the minimization of delay. All the experimental curves we report include 95% bars for the measurement values.

Figure 11 shows the effect of introducing a simple rule that limits the frequency of path switching; each time a source selects a new path identified as causing the smallest delay, the decision is accepted only with probability P (the

Figure 12: Average packet delay vs. switching probability.

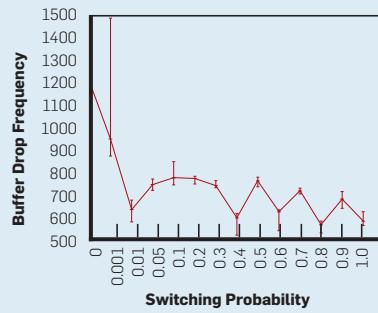


Figure 13: Average packet delay vs. threshold.

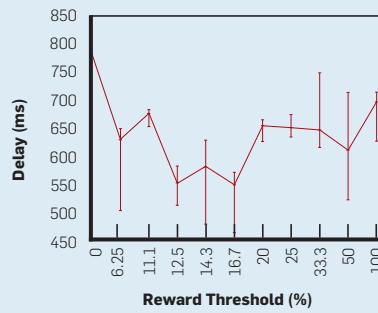


Figure 14: Rate of path oscillations vs. threshold.



switching probability). Thus if $P = 1$ all recommended path switches occur; when $P = 0.001$ only one of every 1,000 recommended path switches actually takes place. Thus the top curve shows how the switching probability affects path oscillations, starting with a given path and returning to it again; as the switching probability increases so does the rate at which paths oscillate. The effect of P on the packet drop rate at the output resequencing buffer is shown in the bottom chart in the figure.

Figure 12 indicates that improvement in QoS (delay in this case) can be achieved with a small switching probability ($P = 0.01$ or a little higher). Path switching improves average delay (and is why CPN attempts to switch paths), though it comes at the cost of packet loss. However, one can mitigate this loss by probabilistically limiting the switching while retaining the benefit of improved QoS by lowering packet delay.

The SAN programmer can also limit oscillations by setting a threshold that allows a path switch only when the projected QoS improvement exceeds the threshold. A small threshold allows more frequent switches and hence potentially more oscillations, but a large threshold may hurt QoS. Figure 13 shows that if the threshold is small, the observed packet delay is large, and as the threshold increases delay improves, but packet delay increases again for larger thresholds. For small threshold values, longer packet delays indicate that switching occurs based on "noise" rather than on real gain. Increasing the threshold in Figure 14 would reduce the oscillations, though the effect would level out quickly. The threshold thus limits the negative effect of switching but preserves the advantages of self-awareness and adaptation.

Conclusion

The approach to developing self-aware networks presented here gives end users the means to explore the state of the network so as to find the best ways to meet their communication needs. Focusing on the primary function of packet routing, I have tried to answer a number of questions concerning the feasibility of such networks and whether reliable communications is possible in largely unknown networks. I have also addressed whether

there is a risk of unstable behavior in such systems due to constant "changes of mind" and oscillations as new information becomes available to users and whether a user's ability to adapt to changing circumstances in the network reduces the consequences of network failure. The experiments reported relate to small (up to 46-node) networks; more results are available at <http://san.ee.ic.ac.uk>.

The Internet consists of hierarchically organized autonomous systems of relatively small size, and one can imagine that routing inside and among them would benefit from the techniques discussed here. Future research is likely to investigate how these ideas can be integrated into existing networks, how they scale to large networks, how they might be able to withstand the malicious behavior of users and network nodes, and how they can support mobile users.

Acknowledgments

Research sponsored by the U.K. Engineering and Physical Sciences Research Council Grant GR/S52360/01 on self-aware networks and quality of service; and by E.U. FP6 Projects on self-aware networks, performance, and adaptivity and componentware for autonomic situation-aware communications and dynamically adaptable services. C

References

- Bennett, J.C.R., Partridge, C., and Shectman, N. Packet reordering is not pathological network behavior. *IEEE/ACM Transactions on Networks* 7, 6 (1999), 789–798.
- Boyan, J.A. and Littman, M.L. Packet routing in dynamically changing networks: A reinforcement learning approach. In *Proceedings of the Advances in Neural Information Processing Systems Conference* (Denver). Morgan Kaufmann, San Francisco, 1994.
- Chen, G., Branch, J., and Szymanski, B.K. A self-selection technique for flooding and routing in wireless ad-hoc networks. *Journal of Network and Systems Management* 14, 3 (2006), 359–380.
- Di Caro, G. and Dorigo, M. Antnet: Distributed stigmergetic control for communication networks. *Journal of AI Research* 9 (1998), 317–365.
- Dobson, S., Denazis, S., Fernández, A., Gaiti, D., Gelenbe, E., Massacci, F., Nixon, P., Saffre, F., Schmidt, N., and Zambonelli, F. A survey of autonomic communications. *ACM Transactions on Autonomous and Adaptive Systems* 1, 2 (2006), 223–259.
- Gao, R., Dovrolis, C., and Zegura, E. Avoiding oscillations due to intelligent route control systems. In *Proceedings of the 25th IEEE International Conference on Computer Communications* (Barcelona, Catalunya, Apr. 23–29). IEEE Press, New York, 2006.
- Gelenbe, E., Sakellari, G., and D'Arizenzo, M. Admission of QoS-aware users in a smart network. *ACM Transactions on Autonomous and Adaptive Systems* 3, 1 (Mar. 2008), 1–28.
- Gelenbe, E. A diffusion model for packet travel time in a random multihop medium. *ACM Transactions on Sensor Networks* 3, 2 (June 2007).
- Gelenbe, E. and Loukas, G. A self-aware approach to denial-of-service defence. *Computer Networks: The International Journal of Computer and Telecommunications Networking* 51, 5 (Apr. 2007), 1299–1314.
- Gelenbe, E., Liu, P., and Lainé, J. Genetic algorithms for route discovery. *IEEE Transactions on Systems, Man, and Cybernetics* B 36, 6 (Dec. 2006), 1247–1254.
- Gelenbe, E., Lent, R., and Nunez, A. Self-aware networks and QoS. *Proceedings of the IEEE* 92, 9 (Sept. 2004), 1478–1489.
- Gelenbe, E. and Lent, R. Power-aware ad hoc cognitive packet networks. *Ad Hoc Networks* 2, 3 (July 2004), 205–216.
- Gelenbe, E., Gellman, M., Lent, R., Liu, P., and Su, P. Autonomous smart routing for network QoS. In *Proceedings of the First International Conference on Autonomous Computing* (May 17–19). IEEE Computer Society Press, New York, 2004, 232–239.
- Gelenbe, E., Lent, R., and Xu, Z. Measurement and performance of a cognitive packet network. *International Journal of Computer and Telecommunications Networking* 37 (2001), 691–791.
- Gelenbe, E. Learning in the recurrent random neural network. *Neural Computation* 5, 1 (1993), 154–164.
- Gellman, M. Oscillations in self-aware networks. *Proceedings of the Royal Society A* 464, 2096 (2008), 2169–2186.
- Keralapura, R., Chuah, C.N., Taft, N., and Iannaccone, G. Can coexisting overlays inadvertently step on each other? In *Proceedings of the 13th IEEE International Conference on Network Protocols* (Boston, Nov. 6–9). IEEE Computer Society, New York, 2005, 201–214.
- Khanna, A. and Zinky, J. The revised Arpanet routing metric. *SIGCOMM Review* 19, 4 (1989), 45–56.
- Koenig, S., Szymanski, B.K., and Liu, Y. Efficient and inefficient ant coverage methods. *Annals of Mathematics and Artificial Intelligence* 31, 1–4 (2001), 41–76.
- Labovitz, C., Malan, G.R., and Jahanian, F. Internet routing instability. *IEEE/ACM Transactions on Networks* 6, 5 (1998), 515–528.
- Liu, P. and Gelenbe, E. Recursive routing in the cognitive packet network. In *Proceedings of the Third International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities* (May 21–23, 2007, Trento, Italy), 21–23.
- Malkin, G. RIP Version 2. RFC 2453, 1998; <http://www.faqs.org/rfcs/rfc2453.html>.
- Moy, J. OSPF Version 2. RFC 2328, 1998; <http://www.ietf.org/rfc/rfc2328.txt>.
- Öke, G.G., Loukas, G., and Gelenbe, E. Detecting denial-of-service attacks with Bayesian classifiers and the random neural network. In *Proceedings of the IEEE International Conference on Fuzzy Systems* (London, 2007), 964–969.
- Papadimitriou, C.H. and Yannakakis, M. Shortest paths without a map. *Theoretical Computer Science* 84, 1 (1991), 127–150.
- Ranadive, U. and Medhi, D. Some observations on the effect of route fluctuation and network link failure on TCP. In *Proceedings of the 10th International Conference on Computer Communications and Networks* (Oct. 15–17, 2001), 460–467.
- Sakellari, G. and Gelenbe, E. Adaptive resilience of the cognitive packet network in the presence of network worms. In *Proceedings of the NATO Symposium on C3I for Crisis, Emergency and Consequence Management* (May 11–12, 2009, Bucharest, Romania).
- Shaikh, A., Varma, A., Kalampoukas, L., and Dube, R. Routing stability in congested networks: Experimentation and analysis. In *Proceedings of SIGCOMM 2000*, (Stockholm, Aug. 29–Sept. 2). ACM Press, New York, 2000, 163–174.
- Thorup, M. and Zwick, U. Compact routing schemes. In *Proceedings of the 13th Annual ACM Symposium on Parallel Algorithms and Architectures* (Heraklion, Crete Island, Greece, July 4–6). ACM Press, New York, 2001, 1–10.

Erol Gelenbe (e.gelenbe@imperial.ac.uk) is Head of the Intelligent Systems and Networks Research Group and Professor in the Dennis Gabor Chair, Electrical and Electronic Engineering Department, Imperial College London.

contributed articles

DOI:10.1145/1538788.1538808

It takes a city of developers to build a big system that is never done.

BY RICK KAZMAN AND HONG-MEI CHEN

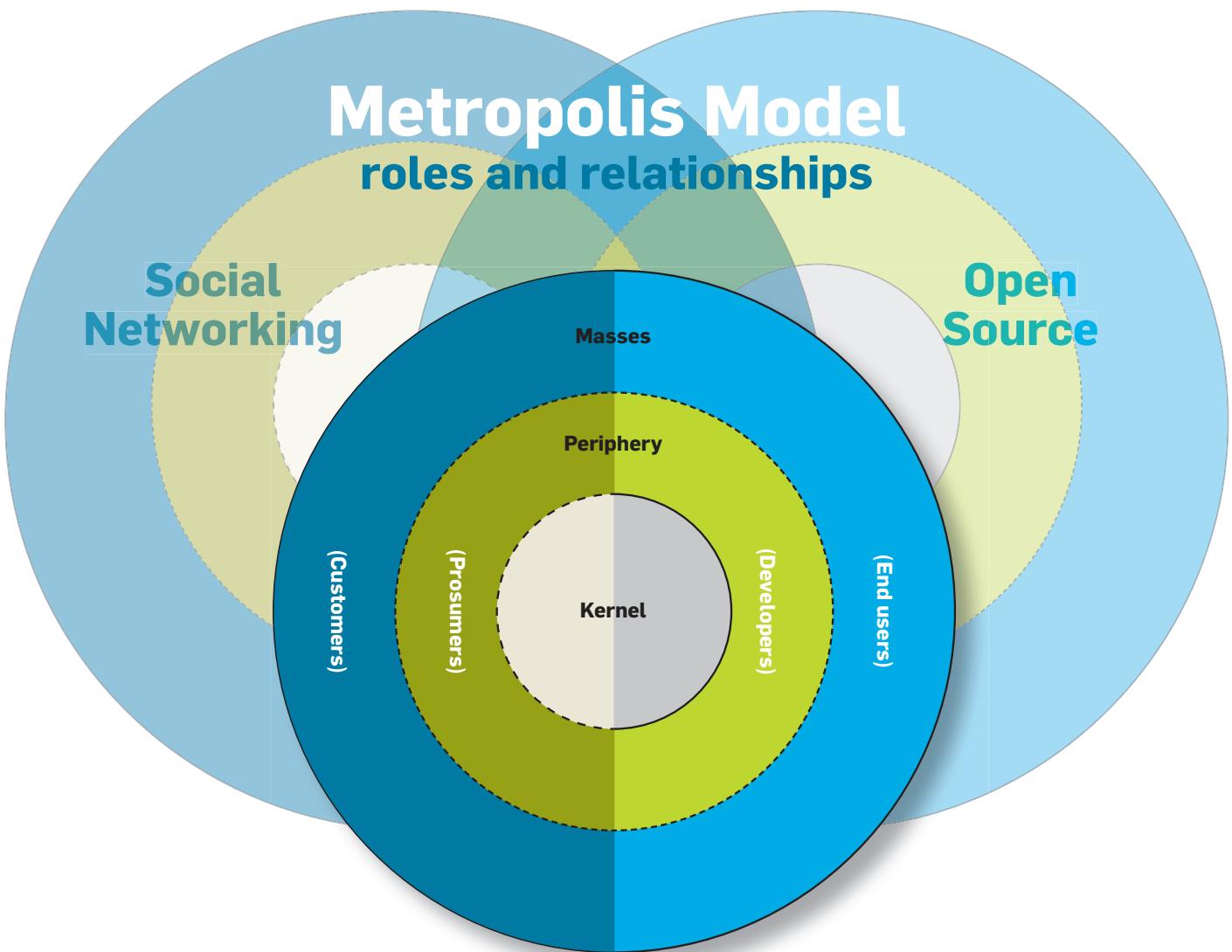
The Metropolis Model A New Logic for Development of Crowdsourced Systems

TWO TRENDS IN business and society are reshaping the world: the rise of the socio-technical network and an emerging service orientation. Benkler⁴ offered a provocative argument about the networked information economy: that we are in the midst of a radical transformation in how we create our information environment. This change is at the heart of the open-source software movement, but OSS is only one example of how society is restructuring around new models of production and consumption of services. The aspect of the restructuring that is most startling “is the rise of effective, large-scale

cooperative efforts—peer production of information, knowledge, and culture...We are beginning to see the expansion of this model not only to our core software platforms, but beyond them into every domain of information and cultural production”⁴ Benkler calls this phenomenon “commons-based peer production,” attributing its rise to the rise of “the network.” The networked information environment has dramatically transformed the marketplace, creating new modes and opportunities for how we produce and consume information. Crowd-sourcing—the popular term for commons-based peer production—is used to create value in information technology, the arts, basic research, and retail business.¹³

A “commons” is the opposite of property, referring rather to a set of shared, accessible community resources. Peer production harnesses the creative energies of many self-selecting participants with little or no financial compensation or formal managerial structure. The importance of this form of production is undeniable; as of May 2009 five of the 10 most popular Web sites—MySpace.com, YouTube.com, Facebook.com, Wikipedia.org, and Blogger.com—were produced this way, according to Alexa.com¹; with the exception of Wikipedia, all are for-profit enterprises.

The second trend, coinciding with and compounding the first, is that organizations are moving toward a service orientation as part of the growing worldwide service economy. Service industries in 2007 accounted for 55% of economic activity in the U.S. (<http://www.census.gov/econ/www/servmenu.html>). Meanwhile, businesses are shifting from a “goods-dominant” view, in which tangible output and discrete transactions are central, to a service-dominant view, in which intangibility, exchange processes, and relationships are central.²⁷ In the old goods-dominant logic, “services” (usually plural) were viewed as either a type of (intangible) good or an add-on that enhanced the



value of a good. In contrast, “service” is now considered “a process of doing something for another party.”²⁷ This service-dominant logic requires a shift on the part of businesses to viewing customers not as passive recipients of goods but as co-creators of value. This implies more than just a move from goods to services but a reframing of the purpose of the enterprise and its role in value creation. The service-dominant perspective has profoundly changed how organizations think about their relationships with their customers—the crowds—and how they leverage them and their resources. This shift in perspective greatly challenges traditional methods of system development.

Traditionally, system analysts are trained to focus on the “value propositions” of firms, not on “value co-creation.” At best, “co-production” with

customers has been used in such design methodologies as Joint Product Design, Joint Application Design, Rapid Application Development, and, more recently, agile methods in which customer requirements are solicited and modeled through an iterative process that incorporates immediate customer feedback. But this still reflects a goods-dominant logic. Product-focused and goods-focused design treats customers as isolated entities—recipients of value—neglecting the customers’ own resources and networks for dynamic collaborative value co-creation. Service-dominant design, on the other hand, considers resource integration from various entities, including customers and firms and their suppliers and networks, for value co-creation.⁶ Examples of co-creation have emerged, from OSS to Wikipedia, Facebook, Amazon’s Me-

chanical Turk, and many other community-based service systems (CBSSs).¹⁵ Each is a complex software-intensive or software-enabled system co-created by its participants—the crowds.⁴

Our existing models of software and system development are of little help in understanding and managing this new form of value co-creation. The older models all contain a “closed world” assumption—that projects have dedicated finite resources, management “manages” these resources, requirements are known, and systems are developed, tested, and released in planned increments. However, these assumptions all break down in a crowd-sourced world.

Here, we offer a set of principles on which a new system-development model—more appropriate for the service-dominant, crowdsourced world—

should be based. We call them the Metropolis Model; metropolis is the Greek word for “city.” The analogy is deliberate; this new form of producing systems is more like constructing a city than a single building, a perspective called ultra-large-scale (ULS).²⁰ ULS systems are like cities in that they are not conceived or built by a single organization, have no centralized control, and are continuously evolving.

The Metropolis Model is our attempt to describe and prescribe the principles surrounding how such systems might be created and sustained. It offers a unified logic for reasoning about and managing system development for the two major forms of crowdsourced systems: OSS development and community-based service systems (see the figure here). A CBSS, which involves creation of content but typically not of software, includes social networking and commercial service systems. The crowds utilized by the two types of systems are also different, as we discuss later. These systems are not new, though their rapid growth and importance is unprecedented. For example, OSS has become an increasingly important sector of the software market; according to a 2008 European Union study of Free/Libre Open Source Software, or FLOSS, the “notional value of Europe’s investment in FLOSS software [represented] 20.5% of total software investment” in 2006.¹¹ This model does not apply to all forms of software creation or system development; some systems are too business-critical, security-critical, or safety-critical to be entrusted to the crowds so will never be produced by a group of peers. But the Metropolis Model clearly applies to a large and fast-growing set of software-centric systems.

Software architects and project managers find it worthwhile to embrace the principles of the Metropolis Model for developing this broad class of systems, taking advantage of the collective wisdom, creativity, and productivity of the crowds. The Model’s principles inevitably lead businesses and project managers to reason differently about virtually every aspect of system development, including project management, requirements elicitation, architecture, implementation, testing, delivery, maintenance, and operations.

One cannot conceive of a crowdsourced system's functionality in terms of "releases" any more than a city has a release.

Characteristics

The Metropolis Model is built on the characteristics of crowdsourced systems, eight of which have been identified (in the ULS report²⁰ and in our own surveys of CBSS and OSS projects) that challenge existing models of system development. They provide the Model’s intellectual motivation. Software and system engineering have long embraced a centralized production model in which requirements are collected and negotiated, projects managed, architectures created, and correctness determined through a controlled, planned process. It is hierarchical and rule-oriented, not commons-based or egalitarian. Even agile methods are centralized, stressing the importance of face-to-face communication and the advantages of the “bullpen,” or open-office environment where workers interact freely.

However, future crowdsourced systems will be community-driven and decentralized, with little overall control, as is the case with CBSS and OSS.¹⁶ Consequently, we can no longer design and implement such systems through older models. Here are the eight characteristics of crowdsourced systems:

Open teams. Assumptions of a closed team of dedicated developers should be abandoned. “Based on our usual assumptions about volunteer projects and decentralized production processes that have no managers, [Linux] was a model that could not succeed. But it did.”⁴ Similarly, the Apache project was not “organized around a single person or primary contributor”⁹ but resulted from a number of Web masters working together, primarily via email. Jimmy Wales, founder of Wikipedia, an example of a CBSS, exercises virtually no control over the community or the ranks of its volunteers.

Mashability. Enormous effort goes into making systems that are difficult to tear apart for historical, intellectual-property, and security reasons. However, “mashability” is a core characteristic of crowdsourced systems. Web browsers make it simple to view any page’s source, and it is accepted practice to use parts of existing Web sites in new creations. For example, Google Maps, prior to making its APIs public, was used in mashups. In Wikipedia, it is accepted and encouraged that articles

do not stand alone, pointing instead to many other articles. Similarly, OSS projects make it easier to create software by composition, as they are “non-rival” (in the economic sense^{3,28})—that is, the consumption of software by one person or project does not make it less available for consumption by another, unlike, say, apples or gasoline, which, once consumed, require additional resources so an equivalent resource may be consumed by another consumer. For example, Linux is a mashup, beginning as an operating system kernel, and owes much of its implementation to a composition with another OSS project—the GNU operating system. This practice of composing OSS is so widespread that the Apache project has created a tool—Maven—for understanding and managing the many transitive dependencies that arise in such projects.¹⁷

Conflicting, unknowable requirements. While iterative life cycles accept that requirements change, they still operate under the assumption that, in any given iteration, a team might still want to collect and analyze these requirements. However, requirements in a crowdsourced system emerge from its participants operating independently. For example, the requirements for the redesigned server architecture and APIs in Apache came from a single core team developer.² Requirements for Wikipedia articles and Facebook applications come from individual authors and application developers. As a consequence, requirements in a crowdsourced system are never globally “knowable” and therefore inevitably overlap or even conflict, just as the requirements of a city’s inhabitants often conflict; for example, some are pro-development, some want more green space, some want public money for public transit, and some want more highways. Many OSS projects use a voting or moderator process to mediate conflicts,¹⁶ but in some crowdsourced systems conflicts (such as similar but competing periphery-created add-ons within Firefox) are simply tolerated.

Continuous evolution. As a consequence of having constantly changing requirements and distributed resources, a crowdsourced system is never “done” and hence never stable. The term “perpetual beta”²¹ describes this new phenomenon. One cannot

conceive of a crowdsourced system’s functionality in terms of “releases” any more than a city has a release. Parts are being created, modified, and torn down at all times. We must accept change as a constant. For example, OSS projects employ a continuous build process,¹⁸ producing a steady stream of incremental releases and relying on the community of users to be part of the quality-assurance process. For example, the Linux mantra is “release early and often.”²³ Iterative and, more recently, agile processes similarly advocate small, frequent releases and tight integration with users. Likewise, on the CBSS side, there is no notion of a release of Wikipedia or Facebook; though the underlying platform for both Web sites has traditional releases, the content is constantly changing.

Focus on operations. Historically, system-development life-cycle models have focused on development and maintenance as the activities of interest. However, much of the value of crowdsourced systems is that they must be as reliable and accessible as a public utility. Many existing crowdsourced systems focus on operations as a core competency,²¹ as in Amazon, eBay, Facebook, Google, Yahoo, and Wikipedia. Downtime for any reason is unacceptable.

Sufficient correctness. Completeness, consistency, and correctness are goals that are, to varying degrees, anathema to crowdsourced systems. The notion of “perpetual beta,” described earlier, is an admission and acceptance of ongoing incompleteness in software.²¹ We are accustomed to a steady stream of releases of our most basic computing infrastructure (such as operating systems, Web browsers, Web servers, and email clients) to address evolving needs, incorporate new features, and correct bugs. Likewise, sufficient correctness is the norm for crowdsourced content. For example, collaborative tagging—enormously valuable for the semantic Web—does not depend on widespread agreement among taggers. Wikipedia never claimed to be complete or even fully correct, though its accuracy has been assessed and found to be similar to the *Encyclopedias Britannica*.¹²

Unstable resources. Peer-produced applications are subject to the whims

of the peers. Resources, including people, computation, information, and connectivity, come and go.¹⁶ Describing OSS development, Mockus et al.,¹⁸ said these systems “are built by potentially large numbers of volunteers...Work is not assigned; people undertake the work they choose to undertake.” However, large numbers tend to ameliorate the whims of any individual or individual resource, while portability of resources has several manifestations:

- In the CBSS arena, large numbers of prosumers (producers who are also consumers of content) make it possible for Wikipedia to be authoritative and users to efficiently download digital content they want through BitTorrent;

- In the computational arena, large numbers of unstable resources result in overall stability and impressive computational power. For example Skype is a threat to traditional phone companies, even though almost all its resources are “contributed” by the masses. Similarly the University of California, Berkeley’s SETI@home project (<http://setiathome.ssl.berkeley.edu/>) has, at times, been rated the most powerful supercomputer in the world, even though it’s powered by “spare” computation from independent contributors; and

- In the OSS arena, large numbers of independent developers working in parallel tend to provide multiple, often overlapping, solutions to a single problem, reducing the importance of the success of any particular solution or individual. The emerging trend is that unstable resources are increasingly accommodated and even embraced as part of the philosophy of building and running crowdsourced systems, even though unstable resources are viewed as anathema to successful projects.

Emergent behaviors. Large-scale systems—computational and biological—exhibit emergent behaviors, a characteristic noted in traffic patterns, epidemics, computer viruses, and systems of systems.¹⁰ Large-scale Web-based applications (such as Second Life, eBay, and MySpace) have certainly seen complex behaviors emerge that are beyond the vision and intent of their creators (such as the “tax revolt” in Second Life and a seller boycott on eBay). Super-linear growth in OSS projects—previously assumed to be im-

possible—appears to be an emergent behavior.^{14,21} Traditional systems have made deterministic behavior a goal. But systems on the Metropolis scale must abandon this assumption; once the crowds are invited in, determinism is lost.

New Logic

These characteristics mean that traditional life-cycle models are inappropriate for describing or developing crowd-sourced systems and thus require a new logic for both development and management. The Metropolis Model captures the characteristics that differentiate crowd-sourced systems, offering a unified view of the two major types of crowd-sourced systems: CBSS and OSS. Unlike traditional system life-cycle models, the Metropolis Model deliberately focuses on the role and nature of creation by crowds.

Different stakeholders have different roles within crowd-sourced systems. For this reason, we distinguish three realms of roles (and associated infrastructure) within a Metropolis Model, as indicated by the “circles”—kernel, periphery, and masses—in the figure. Example roles for people involved in the kernel include architect, business owner, and policy maker; roles at the periphery include developer and prosumer; and roles for the masses include customer and end user.

There are also differences in “permeability” (dashed and solid lines in the figure) between the two major types of crowd-sourced systems. For example, in OSS development it is possible to transition from end user to developer to kernel architect by consistently contributing and moving up through the meritocracy. On the CBSS side, it is generally impossible for a prosumer to be part of the kernel, as a distinct organization typically creates, plans, and manages the kernel.

Principles

Given the fundamental constructs of the Metropolis Model and their associated roles and permeability, we now describe its seven key principles, illustrating how they apply to OSS and CBSS. From them we also develop a set of implications for a new life-cycle model:

Crowd engagement and egalitarian management of open teams. A metropo-

Most important is that the kernel be highly modular, allowing a project to scale as its community grows, while an original visionary developer or team retains intellectual control.

lis without residents and visitors is a ghost town. Absent in prior models, the first and foremost principle of the Metropolis Model is crowd management. Crowds must be engaged for value co-creation. How to engage them is not only a system-level issue but a strategic imperative for businesses. A crowd typically consists of volunteers (hence cheap labor) unknown to the business. As when building a city, infrastructure and rules must be in place to create the social and technical mechanisms needed to engage long-term participation, encouraging community custodianship, recognizing merits of individuals, promoting them through a hierarchy of “ranks” or allowing them to move to a different realm, and finally protecting the community by barring malicious or dangerous participants.

Crowd-management issues overshadow project-management issues (such as cost containment, scheduling, division of labor, and team communication and monitoring) in traditional systems. Focusing on the crowd does not mean that crowd-sourced systems lack traditional cost and scheduling concerns and responsibilities; many do. The main impetus for crowd-sourcing for many organizations is its potential for cost reduction, increased innovation, and quicker development time for delivering products and services that meet customer needs. However, management requirements are totally different. Most important, the management of open teams in the Metropolis Model is not purely, or even primarily, top-down,^{16,18} as discussed earlier. Even though many for-profit companies contribute to OSS projects, the contributions do not change the inherent nature of management in the projects.³ Work is not assigned, and developers largely undertake the work they choose to undertake. Project leaders spend much of their time attracting, motivating, and coordinating a team of talented developers. For example, in OSS projects, there is no project plan, schedule, or list of deliverables.^{16,18} What little management structure exists is based on principles of democracy and, frequently, meritocracy. Kernel team members in OSS projects are typically invited in via a consensus of existing kernel members or some kind of voting

process, but only after first proving themselves in development, debugging, and design. For example, in the Apache project, “Members are people who have contributed for an extended period of time, usually more than six months and are nominated for membership and then voted on by the existing members.”¹⁸ On the CBSS side, Wikipedia contributors are promoted to the rank of “editor” (an unpaid position) only when they receive at least a 75%–80% approval rating from their peers. However, their rights, when it comes to articles, are no different from those of other users. A new user is able to update an article, and no one pulls rank. Wikipedia does have specially elected custodians with the authority to track down and remove privileges from rule violators; the crowds thus assume administrative, promotion, measurement, and asset-protection responsibility.

Bifurcated requirements. Requirements must be bifurcated into:

- ▶ Kernel service that deliver little or no end-user value, as in the Linux kernel, Apache core, Wikipedia wiki, and Facebook application platform; and
- ▶ Periphery contributed by the peer network (the prosumers) that delivers the vast majority of end-user value. Examples include Linux applications and device drivers, Firefox add-ons, Wikipedia articles, and Facebook applications.

The nature of the requirements in these two categories are also different; kernel service requirements concern quality attributes and their trade-offs, while periphery requirements almost exclusively concern end-user perceivable functions. For example, the requirements for Wikipedia’s wiki are totally unrelated to the requirements for Wikipedia’s content. Facebook’s application platform requirements were determined by Facebook (with input from its developers), whereas the requirements for Facebook applications (developed by prosumers) are determined entirely by developers.

Bifurcated architecture. The architecture is divided into a kernel infrastructure and set of peripheral services created by different groups through different processes. Kernel services (such as in Linux, Perl, Apache Core, Wikipedia wiki, and the Facebook application platform) are designed and implemented by a se-

lect set of highly experienced and motivated developers who are themselves users of the product.^{16, 18} These kernel services provide a platform on which subsequent development is based, as in the Linux kernel, along with a set of “zoning rules” (such as the Internet’s communication protocols) or both platform and rules (such as the Facebook application platform). The kernel provides the means for achieving and monitoring quality attributes (such as performance, security, and availability). The architecture of periphery components is enabled and constrained by the kernel through its primitives and compliance with its protocols; the periphery is otherwise unspecified. Each part of the periphery could, in principle, have its own unique architecture. This lack of specification permits unbridled growth and parallel creation at the periphery. Note also that the kernel does not have to be created through a Metropolis life cycle; kernels are created through more conventional means, typically following evolutionary models.

Fragmented implementation. The bifurcation of the kernel and periphery has important consequences for implementation. The vast majority of implementation in the Metropolis Model is crowdsourced, though the crowdsourcing applies only to the periphery. A distinct group implements the kernel, not a crowd but rather a close-knit, highly motivated, coordinated team.¹⁹ As Mockus¹⁸ noted about OSS projects: “Developers are working only on things for which they have a real passion.” The periphery develops at its own pace, to its own standards, using its own tools, releasing code as it pleases. Similarly, in a CBSS, Wikipedia contributors and Facebook application developers contribute their own resources and adhere to no deadlines but their own. There is no overarching plan and no coordination of the activities of the periphery, just as there is no plan for the implementation of a city, which consists of the collective decisions and actions of perhaps millions of homeowners, businesses, contractors, and government organizations. This is different from existing development processes, even distributed development, that assume a central plan, allocation of resources, and schedule to which all distributed participants adhere.

Distributed testing. Verification of the kernel differs from verification of the periphery. Though the kernel must be highly reliable, this requirement is tractable because the kernel is typically small—often orders of magnitude smaller than the periphery—highly controlled, and slow to change. The reliability of the most popular OSS products has been reported to be quite high.¹⁶ The reliability of the periphery is indeterminate; sufficient correctness is the norm. But sufficient correctness is tolerable when the kernel is properly architected, because problems in the periphery do not compromise the kernel. Linus Torvalds, creator of Linux, once said, “When someone sends me patches to do a new filesystem, and I don’t necessarily trust the patches per se, I can still trust the fact that if nobody’s using this file system, it’s not going to impact anything else.” Similarly the Wikipedia wiki is small, heavily tested, and highly reliable. But Wikipedia relies on its distributed network of contributors and editors to vet the accuracy of its prosumer-contributed entries.

Distributed delivery/maintenance. Delivery and maintenance of the kernel differs dramatically from delivery and maintenance of the periphery. The kernel must be stable and when it does change must be backward compatible (such as in terms of Internet protocols and addressing). At the periphery, perpetual beta is the norm, with a constant stream of independent, uncoordinated “releases.” At the periphery, there is no notion of a stable system state. Gradual and fragmented change is typical and expected.⁹

Ubiquitous operations. Metropolis systems are “always on,” even when they’re being upgraded. Complicating this mandate is the fact that upgrades are not ubiquitous; parts of the system at different release levels operate (and interoperate) simultaneously. But for systems built through a Metropolis Model, operations must be a focal activity and, in particular, geared toward ultra-high availability. Also, upgrades must be backward compatible, retaining access to at least kernel functionality, since there is no assumption that all parts of the system will be upgraded at any given point in time. Finally, the ubiquitous-operations principle indi-

cates that Metropolis systems must be able to scale with the number of users; scaling is achieved because the periphery provides its own development and execution resources (such as Skype, BitTorrent, Kazaa, and SETI@home).

Implications

A system-development model is built on a particular logic used to structure, plan, and manage the process of developing a system. The model implies a set of expectations on tools, processes, activities, and roles. Many models (such as waterfall, spiral, and agile) have evolved over the years, each with its own characteristics, strengths, and weaknesses. No one model is best for all projects; each is suited to particular development contexts and characteristics. For instance, agile methods are typically best for projects with rapidly evolving requirements and short time-to-market constraints, whereas a waterfall model is best for large projects with well-understood, stable requirements and complex organizational structures. Accordingly, the Metropolis Model describes a new set of principles and prescribes a new set of activities for an increasingly significant segment of the market—crowdsourced systems, both OSS and CBSS—as we've explored here. The implications of the Metropolis Model force a new perspective on system development in seven important ways:

Focus on crowd management. The Metropolis Model reflects the metaphor of a bull's-eye (as in the figure), as opposed to, say, a waterfall, a spiral, a "V," or other representations adopted by other models. The contrast is salient; the "phases" of development disappear in the bull's-eye. The model focuses managerial attention on the inclusion of customers (the periphery and the masses) for system development, something never previously modeled.

Several challenges arise from the first principle concerning crowd engagement and egalitarian management of open teams for the success of crowdsourced systems. Policies for crowd management must therefore be aligned with an organization's strategic goals and established early. Crowds are good for certain tasks, not for all. Much of the emergent behavior comes from the activity of the crowd. This connection implies that business

models are examined in light of system-development tasks for crowd engagement, performance-management monitoring, and community protection. As crowdsourcing is rooted in the "gift" culture, for-profit organizations must align tasks with volunteers' values and intentions.²⁹ Project managers must set up a management system that is "lightweight" so responsibility for creation is borne by volunteers and capable and robust enough to drive the ongoing success of the site and protect the system from destruction.³⁰

By opening a project to the crowds, management accepts that they consist of unknown people at disparate locations anywhere on the Internet and in time zones, countries, and cultures. This is certainly the case for nontrivial OSS projects. Managing them means the periphery shares in their success and, to a large extent, is self-governing and self-adaptive. Many leaders of important, large-scale open source projects have said they do not "lead" in a traditional sense. For example, Linus Torvalds (creator of Linux) and Larry Wall (creator of Perl) both say they exert no management control and do not command members of the project.³¹ Jimmy Wales (founder of Wikipedia) does not control Wikipedia; indeed, he does not even control the Wikipedia entry on "Jimmy Wales."³² Periphery members cannot be controlled but must be inspired, persuaded, and motivated. Due to its distributed nature, the project must have a clear task breakdown structure but with a minimum of hierarchy and bureaucracy; there must also be collaboration or mass-collaboration technology—typically email lists, wikis, and discussion forums—for communication and coordination.¹⁶ Even the entrance of many for-profit companies into the OSS movement has not changed the inherent nature of project management in these projects, remaining more consensus-based meritocracies than traditional top-down hierarchies.

This culture means management must focus on communication, negotiation, and leadership to guide developers and content creators, persuading them to share in the vision of the project. The creators of the kernel must also commit resources to create effective tutorials and examples. Finally, kernel creators must pay attention to

the usability (simplicity and learnability) of the kernel, making it easy for the periphery to carry on. Wikipedia succeeds, in part, because it is trivial for a prosumer to create or edit an article. Facebook succeeds, in part, because it takes only hours for a developer to create a simple application.

Separate kernel and periphery. The Metropolis Model embeds explicit recognition of separate kernel and periphery in different tools, processes, activities, roles, and expectations for each. The kernel must be small and tightly controlled by a group of developers focusing on modularity, core services, and core quality attributes, enabling unbridled and uncoordinated growth at the periphery.¹⁹ Separation of kernel and periphery is the foundation for the Metropolis Model principles of bifurcated requirements and bifurcated architecture and the foundation for the principles of distributed testing and fragmented implementation.

Change the requirements process. The requirements for Metropolis systems are primarily asserted by the periphery, typically through email, wikis, and discussion forums. These forums must be made available (typically by members of the kernel) and the periphery encouraged to participate in discussions about the requirements to, in effect, create a community. In addition, it must be stressed that these forums are used mainly for discussing the requirements of the core or of significant parts of the periphery. Metropolis projects must, therefore, accept that many requirements for functionality at the periphery may never be discussed. For example, any individual developer may contribute a new device driver to Linux for an obscure device or new Wikipedia entry, and this contribution (and its requirements) might never be discussed in an open forum. This changes the fundamental nature of requirements engineering, which traditionally focuses on collecting requirements, making them complete and consistent, and removing redundancies wherever possible.

Increase attention to architecture. The kernel architecture is the fabric that unites Metropolis systems. As such, it must be designed to accommodate the specific characteristics of CBSSs and OSSs. For this reason, the architecture cannot "emerge," as it often does in

traditional life-cycle and agile models. It must be designed up-front by an experienced, motivated team focusing on modularity to enable the parallel activities of the periphery and the kernel's core quality attributes (such as security, performance, and availability).

A lead architect or small team of leads should be assigned to manage project coordination and have the final say in matters affecting the kernel. For example, Linus Torvalds continues to exert veto rights on matters affecting the Linux kernel. Similarly, members of the team developing Apache's core control changes to the core's architecture, Facebook controls its application platform architecture (even though it is OSS), and Wikipedia.org controls the structure and capabilities of its wiki. Most important is that the kernel be highly modular, allowing a project to scale as its community grows, while an original visionary developer or team retains intellectual control.¹⁸

Plan for distributed testing. Bifurcation of the kernel and periphery provides a guiding principle for testing activities. The kernel must be thoroughly tested and validated, since it unites the system. This imperative can, however, be made tractable. When planning a Metropolis project, project leaders must focus on validation of the kernel and put tools, guidelines, and processes in place to facilitate this validation. For this reason alone the kernel should be kept small. The project should have frequent (perhaps nightly) builds and frequent releases. Bug reporting should be built into the system and require little effort on the part of the periphery. The project should focus on explicitly taking advantage of the "many eyes" touted by OSS development to constantly scrutinize and test the kernel.²³ Such scrutiny does not imply that all aspects of a Metropolis project are thoroughly tested, only that the kernel is.

Create flexible automated delivery mechanisms. Delivery mechanisms must work in a distributed, asynchronous manner and be flexible enough to accept incompleteness of the installed base as the norm. Thus, any delivery mechanism must tolerate older versions, multiple coexisting versions, and even incomplete versions. A Metropolis system should also, as far as

Prior life-cycle models are inadequate—mostly mute—on the concerns of crowdsourcing, super-linear growth, and change as a constant.

possible, be tolerant of incompatibilities in itself and among other systems. For example, modern Web browsers still parse old versions of HTML or interact with old versions of Web servers; add-ons and plug-ins in the Firefox browser coexist at different version levels yet do not "break" the browser. This approach to delivery and maintenance is a direct consequence of the characteristic of sufficient correctness.

Plan for ultra-high availability operation. In most system-development projects, operations are not an early focus of developer attention or resources. In a Metropolis project, the principle of ubiquitous operations must be made a focus due to the distributed and uncoordinated nature of contributions. A Metropolis project must design and plan for ultra-high reliability of the kernel and its infrastructure while paradoxically accepting the fact that periphery software often fails. This focus means the project must explicitly create monitoring mechanisms, determine the current state of the system, and control mechanisms so bugs in the periphery do not undermine the kernel. The project must also avoid any form of centralized critical resources or centralized control—people or computation—as they are potential single points of failure and hence anathema to high availability. In addition, the system must transition smoothly, maintaining continuous operations as it evolves.

Conclusion

Life-cycle models are never revolutionary, arising instead in reaction to ambient conditions in the software-development world. The waterfall model was created almost 40 years ago to focus more attention on removing flaws early in a project's life cycle in reaction to the delays, bugs, and failures of projects of increasing complexity. The spiral model and, later, the Rational Unified Process, were created because projects needed to produce working versions of software more quickly and mitigate risk earlier in the software-development life cycle.⁵ Agile methods grew out of the desire for less bureaucracy, more responsiveness to customer needs, and shorter time to market.

Similarly, the Metropolis Model formally captures a current market response: commons-based peer pro-

duction and service-dominant logic. Prior life-cycle models are inadequate—mostly mute—on the concerns of crowdsourcing, super-linear growth, and change as a constant. While the Metropolis Model is not a life-cycle model, it does offer new ways to think about how a new breed of system can be developed; its principles help management shift to new project-management styles that take advantage of the “wisdom of crowds.” The wrong model or a misaligned model can mean disaster for an organization. The right model—possibly requiring substantial organizational and technological reengineering—provides significant new opportunity. For example, IBM (the most patent-productive company in the world) now makes more money from crowdsourced OSS-related services than from all its patent-protected intellectual property,⁴ even though the shift to OSS was turbulent and controversial within IBM.

The Metropolis Model provides a framework within which organizations are able to reason about all aspects of how they create systems, including tool support, languages, training, resource allocation and management, and personal motivation. The principles of the Metropolis Model are useful as a critical set of questions for examining the alignment of system-development activities with the underlying business model. Business-model questions come first: Who are our customers? What value can be co-created by and for them? What motivation can I offer to engage them for the long term? Answers prompt a new set of system-development questions: How can customer participation be engaged? How can the infrastructure be bifurcated? What technological or system competency must be developed to facilitate engagement and custodianship of the system? What policies must be established to safeguard the community? To answer, organizations must identify the characteristics of their systems and reconsider their business and development models.

Metropolis Model concepts are not appropriate for all forms of development. Smaller systems with limited scope will continue to benefit from the conceptual integrity that accompanies small, cohesive teams. High-security

and safety-critical systems and systems built around protected intellectual property will continue to be built in traditional ways for the foreseeable future. But more and more crowdsourcing, mashups, open source, and other forms of nontraditional development are being harnessed for value co-creation. The Metropolis Model speaks to all of them. For example, mashups are beginning to be observed and supported even in the extremely conservative financial sphere.⁶

Embracing the Metropolis Model requires dramatic changes to accepted software-engineering practices. Organizations must be prepared to adopt new organizational structures, processes, and tools to support these changes. Each Metropolis principle is, to some degree, counterintuitive relative to existing software-engineering practices. Management must therefore guard against old habits and foster a new mindset to deal with unknown people in open teams, embrace incomplete requirements, accept sufficient correctness, and anticipate and tolerate emergent behavior.

Much more research is needed to understand and capitalize on the relatively new form of commons-based peer production. We offer the Metropolis Model as a foundation on which subsequent research and life-cycle models can be built. □

References

1. Alexa.com; http://www.alexa.com/site/ds/top_sites.
2. Apache HTTP Server Project; http://httpd.apache.org/ABOUT_APACHE.html.
3. Barr, J. *The Paradox of Free/Open Source Project Management*; <http://www.linux.com/feature/42466>.
4. Benkler, Y. *The Wealth of Networks: How Social Production Transforms Markets and Freedom*. Yale University Press, New Haven, CT, 2006.
5. Boehm, B. A spiral model of software development and enhancement. *IEEE Computer* 31, 7 (May 1988), 61–72.
6. BusinessWeek.com. E*Trade is banking on Web services (Nov. 13, 2006); http://www.businessweek.com/technology/content/nov2006/tc20061113_151490.htm.
7. Chen, H-M and Vargo, S. Toward an alternate logic for electronic customer relationship management. *International Journal of Business Environment* 2, 2 (2008), 116–132.
8. Cifolilli, A. Phantom authority, self-selective recruitment and retention of members in virtual communities: The case of Wikipedia. *First Monday* 8, 12 (Dec. 2003); <http://firstmonday.org/hbtbin/cgiwrap/bin/ojs/index.php/fm/article/view/1108/1028>.
9. Fielding, R. Shared leadership in the Apache project. *Commun. ACM* 42, 4 (Apr. 1999), 42–43.
10. Fisher, D. *An Emergent Perspective on Interoperation in Systems of Systems*. Technical Report CMU/SEI-2006-TR-003. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2006.
11. Ghosh, R.A., Ed. *Economic Impact of Open Source Software on Innovation and the Competitiveness of the Information and Communication Technologies and safety-critical systems and systems built around protected intellectual property will continue to be built in traditional ways for the foreseeable future. But more and more crowdsourcing, mashups, open source, and other forms of nontraditional development are being harnessed for value co-creation. The Metropolis Model speaks to all of them. For example, mashups are beginning to be observed and supported even in the extremely conservative financial sphere.⁶*
12. Giles, J. Special report: Internet encyclopedias go head to head. *Nature* (Dec. 14, 2005).
13. Howe, J. The rise of crowdsourcing. *Wired* 14, 6 (June 2006); <http://www.wired.com/wired/archive/14.06/crowds.html>.
14. Koch, S. Software evolution in open source projects: A large-scale investigation. *Journal of Software Maintenance and Evolution: Research and Practice* 19, 6 (Nov./Dec. 2007), 361–382.
15. Maglio, P., Srinivasan, S., Kreulen, J., and Spohrer, J. Service systems, service scientists, SSME, and innovation. *Commun. ACM* 49, 7 (July 2006), 81–85.
16. Markus, M.L., Manville, B., and Agres, C. What makes a virtual organization work?. *Sloan Management Review* (Fall 2000), 13–25.
17. Maven Project; <http://maven.apache.org/index.html>.
18. Mockus, A., Fielding, R., and Herbsleb, J. Two case studies of open source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology* 11, 3 (July 2002), 309–346.
19. Narduzzo, A. and Rossi, A. *Modularity in Action: GNU/Linux and Free/Open Source Software Development Model Unleashed*. Research on Organizations, Coordination & Knowledge working papers 020, Dept. of Computer and Management Sciences, University of Trento, Italy, 2003.
20. Northrop, L., Feiler, P., Gabriel, R., Goodenough, J., Linger, R., Longstaff, T., Kazman, R., Klein, M., Schmidt, D., Sullivan, K., and Wallnau, K. *Ultra-Large-Scale Systems: The Software Challenge of the Future*. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2006.
21. O'Reilly, T. What is Web 2.0? Design patterns and business models for the next generation of software; <http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html>.
22. O'Reilly, T. Lessons from open-source software development. *Commun. ACM* 42, 4 (Apr. 1999), 33–37.
23. Raymond, E.S. *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. O'Reilly & Associates, Sebastopol, CA, 1999.
24. Scacchi, W. Understanding the requirements for developing open source software systems. *IEEE Proceedings Software* 149, 1 (Feb. 2002), 24–39.
25. Scacchi, W. Free/open source software development: Recent research results and emerging opportunities. In *Proceedings of the Sixth Joint Meeting of the European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering* (Dubrovnik, Croatia, Sept. 3–7). ACM Press, New York, 2007, 459–468.
26. Torvalds, L. The Linux edge. In *Open Sources: Voices from the Open Source Revolution*, C. DiBona, S. Ockman, and M. Stone, Eds. O'Reilly & Associates, Sebastopol, CA, 1999.
27. Vargo, S. and Lusch, R. Evolving to a new dominant logic for marketing. *Journal of Marketing* 68 (Jan. 2004), 1–17.
28. von Hippel, E. and von Krogh, G. Open source software and the ‘private-collective’ innovation model: Issues for organization science. *Organization Science* 14, 2 (Mar.–Apr. 2003), 209–223.
29. Wagner, C. and Majchrzak, A. Enabling customer-centricity—using wikis and the wiki way. *Journal of Management Information Systems* 23, 3 (Winter 2006–7), 17–43.
30. Wikipedia.org. Jimmy Wales; http://en.wikipedia.org/wiki/Jimmy_Wales#Editing_of_own_Wikipedia_biography.

Sector in the E.U. White paper, Nov. 2006; <http://ec.europa.eu/enterprise/ict/policy/doc/2006-11-20-flossimpact.pdf>.

12. Giles, J. Special report: Internet encyclopedias go head to head. *Nature* (Dec. 14, 2005).
13. Howe, J. The rise of crowdsourcing. *Wired* 14, 6 (June 2006); <http://www.wired.com/wired/archive/14.06/crowds.html>.
14. Koch, S. Software evolution in open source projects: A large-scale investigation. *Journal of Software Maintenance and Evolution: Research and Practice* 19, 6 (Nov./Dec. 2007), 361–382.
15. Maglio, P., Srinivasan, S., Kreulen, J., and Spohrer, J. Service systems, service scientists, SSME, and innovation. *Commun. ACM* 49, 7 (July 2006), 81–85.
16. Markus, M.L., Manville, B., and Agres, C. What makes a virtual organization work?. *Sloan Management Review* (Fall 2000), 13–25.
17. Maven Project; <http://maven.apache.org/index.html>.
18. Mockus, A., Fielding, R., and Herbsleb, J. Two case studies of open source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology* 11, 3 (July 2002), 309–346.
19. Narduzzo, A. and Rossi, A. *Modularity in Action: GNU/Linux and Free/Open Source Software Development Model Unleashed*. Research on Organizations, Coordination & Knowledge working papers 020, Dept. of Computer and Management Sciences, University of Trento, Italy, 2003.
20. Northrop, L., Feiler, P., Gabriel, R., Goodenough, J., Linger, R., Longstaff, T., Kazman, R., Klein, M., Schmidt, D., Sullivan, K., and Wallnau, K. *Ultra-Large-Scale Systems: The Software Challenge of the Future*. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2006.
21. O'Reilly, T. What is Web 2.0? Design patterns and business models for the next generation of software; <http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html>.
22. O'Reilly, T. Lessons from open-source software development. *Commun. ACM* 42, 4 (Apr. 1999), 33–37.
23. Raymond, E.S. *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. O'Reilly & Associates, Sebastopol, CA, 1999.
24. Scacchi, W. Understanding the requirements for developing open source software systems. *IEEE Proceedings Software* 149, 1 (Feb. 2002), 24–39.
25. Scacchi, W. Free/open source software development: Recent research results and emerging opportunities. In *Proceedings of the Sixth Joint Meeting of the European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering* (Dubrovnik, Croatia, Sept. 3–7). ACM Press, New York, 2007, 459–468.
26. Torvalds, L. The Linux edge. In *Open Sources: Voices from the Open Source Revolution*, C. DiBona, S. Ockman, and M. Stone, Eds. O'Reilly & Associates, Sebastopol, CA, 1999.
27. Vargo, S. and Lusch, R. Evolving to a new dominant logic for marketing. *Journal of Marketing* 68 (Jan. 2004), 1–17.
28. von Hippel, E. and von Krogh, G. Open source software and the ‘private-collective’ innovation model: Issues for organization science. *Organization Science* 14, 2 (Mar.–Apr. 2003), 209–223.
29. Wagner, C. and Majchrzak, A. Enabling customer-centricity—using wikis and the wiki way. *Journal of Management Information Systems* 23, 3 (Winter 2006–7), 17–43.
30. Wikipedia.org. Jimmy Wales; http://en.wikipedia.org/wiki/Jimmy_Wales#Editing_of_own_Wikipedia_biography.

Rick Kazman (kazman@hawaii.edu) is a professor in the Department of Information Technology Management in the Shidler College of Business at the University of Hawaii, Honolulu, HI, and a visiting scientist at the Software Engineering Institute of Carnegie Mellon University, Pittsburgh, PA.

Hong-Mei Chen (hmchen@hawaii.edu) is a professor of IT management in the Department of Information Technology Management in the Shidler College of Business at the University of Hawaii at Manoa, Honolulu, HI.

The Best Place to Find the Perfect Job... Is Just a Click Away!

No need to get lost on commercial job boards.
The ACM Career & Job Center is tailored specifically for you.

JOBSEEKERS

- ❖ Manage your job search
- ❖ Access hundreds of corporate job postings
- ❖ Post an anonymous resume
- ❖ Advanced Job Alert system

EMPLOYERS

- ❖ Quickly post job openings
- ❖ Manage your online recruiting efforts
- ❖ Advanced resume searching capabilities
- ❖ Reach targeted & qualified candidates

NEVER LET A JOB OPPORTUNITY PASS YOU BY!
START YOUR JOB SEARCH TODAY!

<http://www.acm.org/careercenter>



Association for
Computing Machinery

Advancing Computing as a Science & Profession



POWERED BY JOBTARGET

review articles

DOI:10.1145/1538788.1538810

Treasures abound from hidden facts found in imprecise data sets.

BY NILESH DALVI, CHRISTOPHER RÉ, AND DAN SUCIU

Probabilistic Databases: Diamonds in the Dirt

A WIDE RANGE of applications have recently emerged that require managing large, imprecise data sets. The reasons for imprecision in data are as diverse as the applications themselves: in sensor and RFID data, imprecision is due to measurement errors,^{15, 34} in information extraction, imprecision comes from the inherent ambiguity in natural-language text,^{20, 26} and in business intelligence, imprecision is tolerated because of the high cost of data cleaning.⁵ In some applications, such as privacy, it is a requirement that the data be less precise. For example, imprecision is purposely inserted to hide sensitive attributes of individuals so that the data may be published.³⁰ Imprecise data has no place in traditional, precise database applications like payroll and inventory, and so, current database management systems are not prepared to deal with it. In contrast, the newly emerging applications offer value precisely because

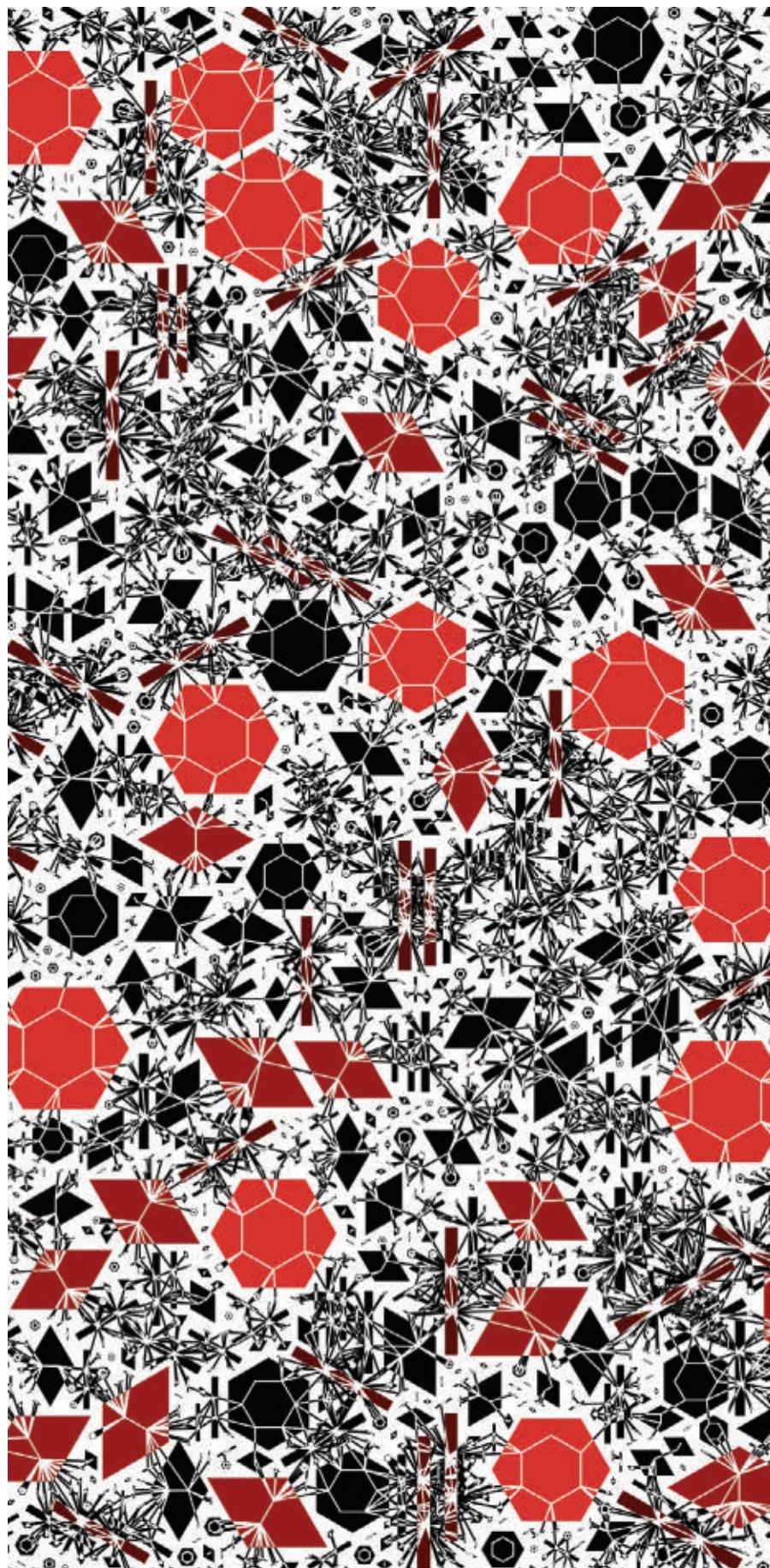
they query, search, and aggregate large volumes of imprecise data to find the “diamonds in the dirt.” This wide variety of new applications points to the need for generic tools to manage imprecise data. In this article, we survey the state of the art of techniques that handle imprecise data by modeling it as probabilistic data.^{2–4, 7, 12, 15, 23, 27, 36}

A *probabilistic database management system*, or PROBDMS, is a system that stores large volumes of probabilistic data and supports complex queries. A PROBDMS may also need to perform some additional tasks, such as updates or recovery, but these do not differ from those in conventional database management systems and will not be discussed here. The major challenge in a PROBDMS is that it needs both to *scale* to large data volumes, a core competence of database management systems, and to do *probabilistic inference*, which is a problem studied in AI. While many scalable data management systems exist, probabilistic inference is a hard problem,³⁵ and current systems do not scale to the same extent as data management systems do. To address this challenge, researchers have focused on the specific nature of relational probabilistic data, and exploited the special form of probabilistic inference that occurs during query evaluation. A number of such results have emerged recently: lineage-based representations,⁴ safe plans,¹¹ algorithms for top-k queries,^{31, 37} and representations of views over probabilistic data.³³ What is common to all these results is they apply and extend well-known concepts that are fundamental to data management, such as the separation of query and data when analyzing complexity,³⁸ incomplete databases,²² the threshold algorithm,¹⁶ and the use of materialized views to answer queries.²¹ In this article, we briefly survey the key concepts in probabilistic database systems and explain the intellectual roots of these concepts in data management.

An Example: The Purple Sox System

We illustrate using an example from an information extraction system. The Purple Sox^a system at Yahoo! Research focuses on technologies to extract and manage structured information from the Web related to a specific community. An example is the DbLife system¹⁴ that aggregates structured information about the database community from data on the Web. The system extracts lists of database researchers together with structured, related information such as publications they authored, their coauthor relationships, talks they have given, their current affiliations, and their professional services. Figure 1(a) illustrates the researchers' affiliations, and Figure 1(b) illustrates their professional activities. Although most researchers have a single affiliation, in the data in Figure 1(a), the extracted affiliations are not unique. This occurs because outdated/errorneous information is often present on the Web, and even if the extractor is operating on an up-to-date Web page, the difficulty of the extraction problem forces the extractors to produce many alternative extractions or risk missing valuable data. Thus, each Name contains several possible affiliations. One can think of Affiliation as being an attribute with uncertain values. Equivalently, one can think of each row as being a separate uncertain tuple. There are two constraints on this data: tuples with the same Name but different Affiliation are mutually exclusive; and tuples with different values of Name are independent. The professional services shown in Figure 1(b) are extracted from conference Web pages, and are also imprecise: in our example, each record in this table is an independent extraction and assumed to be independent.

In both examples, the uncertainty in the data is represented as a probabilistic confidence score, which is computed by the extractor. For example, Conditional Random Fields produce extractions with semantically meaningful confidence scores.²⁰ Other sources of uncertainty can also be converted to confidence scores, for example, probabilities produced by *entity matching algorithms*. (Does the men-



^a <http://research.yahoo.com/node/498>

tioned *Fred* in one Web page refer to the same entity as *Fred* in another Web page?) The example in Figure 1 presents a very simplified view of a general principle: uncertain data is annotated with a confidence score, which is interpreted as a probability. Here, we use “probabilistic data” and “uncertain data” as synonyms.

Facets of a PROBDMS

There are three important, related facets of any PROBDMS: How do we store (or represent) a probabilistic database? How do we answer queries using our chosen representation? How do we present the result of queries to the user?

There is a tension between the power of a representation system, that is, as the system more faithfully models correlations, it becomes increasingly difficult to scale the system. A simple representation where each tuple is an independent probabilistic event is easier to process, but it cannot faithfully model the correlations important to all applications. In contrast, a more complicated representation, for example, a large Markov Network,⁹ can capture the semantics of the data very faithfully, but it may be impossible to compute even simple SQL queries using this representation. An extra challenge is to ensure the representation system maps smoothly to relational data, so that the nonprobabilistic part of the data can be processed by a conventional database system.

A PROBDMS must support complex, decision-support style SQL, with ag-

gregates. While some applications can benefit from point queries, the real value comes from queries that search many tuples, or aggregate over many data values. For example, the answer to *find the affiliation of PC Chair of SIGMOD'2008* is inherently imprecise (and can be answered more effectively by consulting the SIGMOD'2008 home page), but a query like *find all institutions (affiliations) with more than 20 SIGMOD and VLDB PC Members* returns more interesting answers. There are two logical steps in computing an SQL query on probabilistic data: first, fetch and transform the data, and second, perform probabilistic inference. A straightforward but naïve approach is to separate the two steps: use a database engine for the first step and a general-purpose probabilistic inference technique^{9,13} for the second. But on large data the probabilistic inference quickly dominates the total running time. A better approach is to integrate the two steps, which allows us to leverage some database-specific techniques, such as query optimization, using materialized views, and schema information, to speed up the probabilistic inference.

Designing a good user interface raises new challenges. The answer to an SQL query is a set of tuples, and it is critical to find some way to rank these tuples because there are typically lots of false positives when the input data is imprecise. Alternatively, aggregation queries can extract value from imprecise data, because errors tend to cancel each other out (the Law of the Large

Numbers). A separate and difficult task is how to indicate to the user the correlations between the output tuples. For example, the two highest ranked tuples may be mutually exclusive, but they could also be positively correlated. As a result, their ranks alone convey insufficient information to the user. Finally, a major challenge of this facet is how to obtain feedback from the users and how to employ this feedback to “clean” the underlying database. This is a hard problem, that to date has not yet been solved.

Probabilistic databases have found usage in a wide class of applications. *Sensor data* is obtained from battery-powered sensors that acquire temperature, pressure, or humidity readings from the surrounding environment. The BBQ system¹⁵ showed that a probabilistic data model could represent well this kind of sensor data. A key insight was the probabilistic model could answer many queries with sufficient confidence without needing to acquire additional readings. This is an important optimization since acquiring fewer sensor readings allows longer battery life, and so more longer lasting sensor deployments. *Information Extraction* is a process that extracts data items of a given type from large corpora of text.²⁶ The extraction is always noisy, and the system often produces several alternatives. Gupta and Sarawagi²⁰ have argued that such data is best stored and processed by a probabilistic database. In *Data Cleaning*, deduplication is one of the key components and is also a noisy and imperfect process.

Figure 1: Example of a probabilistic database. This is a block-independent-disjoint database: the eight tuples in Researchers are grouped in four groups of disjoint events, for example, t_1^1, t_1^2, t_1^3 are disjoint, and so are, t_4^1, t_4^2 , while tuples from different blocks are independent, for example, t_1^1, t_2^1, t_4^1 are independent; the five tuples in Services are independent probabilistic events. This database can be represented as a c-table using the hidden variables X_1, X_2, X_3, X_4 for Researchers and Y_1, Y_2, Y_3, Y_4, Y_5 for Services.

Researchers:

Name	Affiliation	P
t_1^1	U. Washington	$p_1^1 = 0.3$
	U. Wisconsin	$p_1^2 = 0.2$
	Y! Research	$p_1^3 = 0.5$
t_2^1	U. Washington	$p_2^1 = 1.0$
	U. Wisconsin	$p_3^1 = 0.7$
	U. Washington	$p_3^2 = 0.3$
t_4^1	Y! Research	$p_4^1 = 0.9$
	M. Research	$p_4^2 = 0.1$

(a)

Services:

Name	Conference	Role	P
Fred	VLDB	Session Chair	$q_1 = 0.2$
	VLDB	PC Member	$q_2 = 0.8$
John	SIGMOD	PC Member	$q_3 = 0.7$
	VLDB	PC Member	$q_4 = 0.7$
Sue	SIGMOD	Chair	$q_5 = 0.5$

(b)

Andritsos, Fuxman, and Miller¹ have shown that a probabilistic database can simplify the deduplication task, by allowing multiple conflicting tuples to coexist in the database. Many other applications have looked at probabilistic databases for their data management needs: RFID data management,³⁴ management of *anonymized data*,³⁰ and scientific data management.²⁸

We present a number of key concepts for managing probabilistic data that have emerged in recent years. We group these concepts by the three facets, although some concepts may be relevant to more than one facet.

Facet 1: Semantics and Representation

The de facto formal semantics of a probabilistic database is the *possible worlds model*.¹² By contrast, there is no agreement on a representation system, instead there are several approaches covering a spectrum between expressive power and usability.⁴ A key concept in most representation systems is that of *lineage*, which is derived from early work on incomplete databases by Imielinski and Lipski.²²

Possible Worlds Semantics. In its most general form, a probabilistic database is a probability space over the possible contents of the database. It is customary to denote a (conventional) relational database instance with the letter I . Assuming there is a single table in our database, I is simply a set of tuples (records) representing that table; this is a conventional database. A *probabilistic database* is a discrete probability space $PDB = (W, P)$, where $W = \{I_1, I_2, \dots, I_n\}$ is a set of possible instances, called *possible worlds*, and $P: W \rightarrow [0, 1]$ is such that $\sum_{j=1, n} P(I_j) = 1$. In the terminology of networks of belief, there is one random variable for each possible tuple whose values are 0 (meaning that the tuple is not present) or 1 (meaning that the tuple is present), and a probabilistic database is a joint probability distribution over the values of these random variables.

This is a very powerful definition that encompasses all the concrete data models over discrete domains that have been studied. In practice, however, one must step back from this generality and impose some workable restrictions, but it is always helpful to

All representation formalisms are, at their core, an instance of database normalization: they decompose a probabilistic database with correlated tuples into several BID tables.

keep the general model in mind. Note that in our discussion we restrict ourselves to discrete domains: although probabilistic databases with continuous attributes are needed in some applications,^{7, 15} no formal semantics in terms of possible worlds has been proposed so far.

Consider some tuple t (we use interchangeably the terms *tuple* and *record* in this article). The probability that the tuple belongs to a randomly chosen world is $P(t) = \sum_{j: t \in I_j} P(I_j)$, and is also called the marginal probability of the tuple t . Similarly, if we have two tuples t_1, t_2 , we can examine the probability that *both* are present in a randomly chosen world, denoted $P(t_1 t_2)$. When the latter is $P(t_1)P(t_2)$, we say that t_1, t_2 are independent tuples; if it is 0 then we say that t_1, t_2 are disjoint tuples or exclusive tuples. If none of these hold, then the tuples are correlated in a nonobvious way. Consider a query Q , expressed in some relational query language like SQL, and a possible tuple t in the query's answer. $P(t \in Q)$ denotes the probability that, in a randomly chosen world, t is an answer to Q . The job of a probabilistic database system is to return all possible tuples t_1, t_2, \dots together with their probabilities $P(t_1 \in Q), P(t_2 \in Q), \dots$

Representation Formalisms. In practice, one can never enumerate all possible worlds, and instead we need to use some more concise representation formalism. One way to achieve that is to restrict the class of probabilistic databases that one may represent. A popular approach is to restrict the possible tuples to be either independent or disjoint. Call a probabilistic database *block independent-disjoint*, or *BID*, if the set of all possible tuples can be partitioned into blocks such that tuples from the same block are disjoint events, and tuples from distinct blocks are independent. A BID database is specified by defining the partition into blocks, and by listing the tuples' marginal probabilities. This is illustrated in Figure 1. The blocks are obtained by grouping Researchers by Name, and grouping Services by (Name, Conference, Role). The probabilities are given by the P attribute. Thus, the tuples t_1^2 and t_1^3 are disjoint (they are in the same block), while the tuples t_1^1, t_2^2, s_1, s_2 are independent (they are from different blocks). An intuitive BID model was

Figure 2: An example of a c-table.

Location
U. Washington
U. Wisconsin
Y! Research

$(X_1 = 1) \wedge (Y_1 = 1) \vee (X_1 = 1) \wedge (Y_2 = 1) \vee (X_3 = 2) \wedge (Y_4 = 1)$
 $(X_1 = 2) \wedge (Y_1 = 1) \vee (X_1 = 2) \wedge (Y_2 = 1) \vee (X_3 = 1) \wedge (X_4 = 1)$
 $(X_1 = 3) \wedge (Y_1 = 1) \vee (X_1 = 3) \wedge (Y_2 = 1)$

introduced by Trio⁴ and consists of *maybe-tuples*, which may or may not be in the database, and *x-tuples*, which are sets of mutually exclusive tuples.

Several applications require a richer representation formalism, one that can express complex correlations between tuples, and several such formalisms have been described in the literature: lineage-based,^{4, 18} U-relations,² or the closure of BID tables under conjunctive queries.¹² Others are the Probabilistic Relational Model of Friedman et al.¹⁷ that separates the data from the probabilistic network, and Markov Chains.^{25, 34} Expressive formalisms, however, are often difficult to understand by users, and increase the complexity of query evaluation, which lead researchers to search for simpler, *workable* models for probabilistic data.⁴

All representation formalisms are, at their core, an instance of database normalization: they decompose a probabilistic database with correlated tuples into several BID tables. This is similar to the factor decomposition in graphical models,⁹ and also similar to database normalization based on multivalued dependencies.³⁹ A first question is how to design the normal representation given a probabilistic database. This requires a combination of techniques from graphical models and database normalization, but, while the connection between these two theories was described by Verma and Pearl³⁹ in the early 1990s, to date there exists no comprehensive theory of normalization for probabilistic databases. A second question is how to recover the complex probabilistic database from its normalized representation as BID tables. This can be done through SQL views¹² or through lineage.

Lineage. The *lineage* of a tuple is an annotation that defines its derivation. Lineage is used both to represent probabilistic data, and to represent query results. The Trio system⁴ recognized the importance of lineage in managing

data with uncertainty, and called itself a ULDB, for *uncertainty-lineage database*. In Trio, when new data is produced by queries over uncertain data, the lineage is computed automatically and captures all correlations needed for computing subsequent queries over the derived data.

Lineage also provides a powerful mechanism for understanding and resolving uncertainty. With lineage, user feedback on correctness of results can be traced back to the sources of the relevant data, allowing unreliable sources to be identified. Users can provide much detailed feedback if data lineage is made visible to them. For example, in information extraction applications where data items are generated by pipelines of AI operators, users can indicate if a data item is correct, as well as look at the lineage of data items to locate the exact operator making the error.

The notion of lineage is derived from a landmark paper by Imielinski and Lipski²² from 1984, who introduced *c-tables* as a formalism for representing incomplete databases. We describe c-tables and lineage by using the example in Figure 2. In a c-table, each tuple is annotated with a Boolean expression over some hidden variables; today, we call that expression *lineage*. In our example there are three tuples, U. of Washington, U. of Wisconsin, and Y! Research, each annotated with a lineage expression over variables X_1, X_3, Y_1, Y_2, Y_4 . The semantics of a c-table is a set of possible worlds. An assignment of the variables defines the world consisting of precisely those tuples whose lineage is true under that assignment, and the c-table “means” the set of possible worlds defined by all possible assignments. For an illustration, in our example any assignment containing $X_1 = 3, Y_2 = 1, X_3 = 2, Y_4 = 1$ (and any values for the other variables) defines the world {Y! Research, U. of Washington}, while any assignment with $Y_1 = Y_2 = Y_3 = 0$ defines the empty world.

Lineage is a powerful tool in PROBDMS because of the following important property: the answer to a query over a c-table can always be represented as another c-table, using the same hidden variables. In other words, it is always possible to compute the lineage of the output tuples from those of the input tuples. This is called a *closure property* and was first shown by Imielinski and Lipski.²² We illustrate this property on the database in Figure 1, where each tuple has a very simple lineage. Consider now the SQL query in Figure 3(a), which finds the affiliations of all people who performed some service for the VLDB conference. The answer to this query is precisely the c-table in Figure 2.

Facet 2: Query Evaluation

Query evaluation is the most difficult technical challenge in a PROBDMS. One approach is to separate the query and lineage evaluation from the probabilistic inference on the lineage expression. Various algorithms have been used for the latter, such as Monte Carlo approximation algorithms.^{11, 31} Recently, a much more general Monte Carlo framework has been proposed by Jampani et al.²³ Variable Elimination⁹ was used by Sen and Deshpande.³⁶

Another approach is to integrate the probabilistic inference with the query computation step. With this approach, one can leverage standard data management techniques to speed up the probabilistic inference, such as static analysis on the query or using materialized views. This has led to safe queries and to partial representations of materialized views.

Safety. Two of the authors showed that certain SQL queries can be evaluated on a probabilistic database by pushing the probabilistic inference completely inside the query plan.¹¹ Thus, for these queries there is no need for a separate probabilistic inference step: the output probabilities are computed inside the database engine, during normal query processing. The performance improvements can be large, for example, Ré et al.³¹ observed two orders of magnitude improvements over a Monte Carlo simulation. Queries for which this is possible are called *safe queries*, and the relational plan that computes the output probabilities

correctly is called a *safe plan*. To understand the context of this result we review a fundamental principle in relational query processing: the separation between *what* and *how*.

In a relational query the user specifies *what* she wants: relational query languages like SQL are declarative. The system translates the query into relational algebra, using operators like join \bowtie , selection σ , projection-with-duplicate-elimination Π . The resulting expression is called a *relational plan* and represents *how* the query will be evaluated. The separation between *what* and *how* was first articulated by Codd when he introduced the relational data model,⁸ and is at the core of any modern relational database system. A *safe plan* allows probabilities to be computed in the relational algebra, by extending its operators to manipulate probabilities.¹² There are multiple ways to extend them, the simplest is to assume all tuples to be independent: a join that combines two tuples computes the new probability as $p_1 p_2$, and a duplicate elimination that replaces n tuples with one tuple computes the output probability as $1 - (1 - p_1) \dots (1 - p_n)$. A *safe plan* is by definition a plan in which all these operations are provably correct. The correctness proof (or safety property) needs to be done by the query optimizer, through a static analysis on the plan. Importantly, safety does not depend on the actual instance of the database, instead, once a plan has been proven to be safe, it can be executed on any database instance.

We illustrate with the query in Figure 3(a). Any modern relational database engine will translate it into the logical plan shown in (b). However, this plan is not safe, because the operation $\Pi_{\text{Affiliation}}$ (projection-with-duplicate-elimination) combines tuples that are not independent, and therefore the output probabilities are incorrect. The figure illustrates this for the output value $Y! \text{Research}$, by tracing its computation through the query plan: the output probability is $1 - (1 - p_3 q_1)(1 - p_3 q_2)$. However, the lineage of $Y! \text{Research}$ is $(X_1 = 3 \wedge Y_1 = 1) \wedge (X_1 = 3 \wedge Y_2 = 1)$, hence the correct probability is $p_3(1 - (1 - q_1)(1 - q_2))$.

Alternatively, consider the plan shown in (c). This plan performs an early projection and duplicate elimination

on Services. It is logically equivalent to the plan in (b), i.e., the extra duplicate elimination is harmless. However, the new plan computes the output probability correctly: the figure illustrates this for the same output value, $Y! \text{Research}$. Note that although plans (b) and (c) are logically equivalent over conventional databases, they are no longer equivalent in their treatment of probabilities: one is safe, the other not.

Safety is a central concept in query processing on probabilistic databases. A query optimizer needs to search not just for a plan with lowest cost, but for one that is safe as well, and this may affect the search strategy and the outcome: in a conventional database there is no reason to favor the plan in (c) over that in (b) (and in fact modern optimizers will not choose plan (c) because the extra duplication elimination increases the cost), but in a probabilistic database plan (c) is safe while (b) is unsafe. A safe plan can be executed directly by a database engine with only small changes to the implementation of its relational operators. Alternatively, a safe plan can be executed by expressing it in regular SQL and executing it on a conventional database engine, without any changes: Figure 3(d) illustrates how the safe plan can be converted back into SQL.

Safe plans have been described for databases with independent tuples,¹¹ for BID databases,¹² for queries whose predicates have aggregate operators,³² and for Markov Chain databases.³⁴ While conceptually a safe plan ties the probabilistic inference to the query plan, Olteanu et al.²⁹ have shown that it is possible to separate them at runtime: the optimizer is free to choose any query plan (not necessarily safe), then the probabilistic inference is guided by the information collected from the safe plan. This results in significant execution speed-up for typical SQL queries.

Dichotomy of Query Evaluation. Unfortunately, not all queries admit safe plans. In general, query evaluation on a probabilistic database is no easier than general probabilistic inference. The latter is known to be a hard problem.³⁵ In databases, however, one can approach the query evaluation problem differently, in a way that is best explained by recalling an important

distinction made by Vardi in a landmark paper in 1982.³⁸ He proposed that the query expression (which is small) and the database (which is large) be treated as two different inputs to the query evaluation problem, leading to three different complexity measures: the data complexity (when the query is fixed), the expression complexity (when the database is fixed), and the combined complexity (when both are part of the input). For example, in conventional databases, all queries have data complexity in PTIME, while the combined complexity is PSPACE complete.

We apply the same distinction to query evaluation on probabilistic databases. Here the data complexity offers a more striking picture: some queries are in PTIME (for example, all safe queries), while others have #P-hard data complexity. In fact, for certain query languages or under certain assumptions it is possible to prove a complete dichotomy, that is, each query belongs to one of these two classes.^{10, 12, 32, 34} Figure 4 describes the simplest dichotomy theorem, for conjunctive queries without self-joins over databases with independent tuples, first proven by Dalvi and Suciu.¹¹ Safe queries are by definition in the first class; under the dichotomy property, any unsafe query has #P-hard data complexity. For unsafe queries we really have no choice but to resort to a probabilistic inference algorithm that solves, or approximates, a #P-hard problem. The abrupt change in complexity from PTIME to #P-hard is unique to probabilistic databases, and it means that query optimizers need to make special efforts to identify and use safe queries.

An active line of research develops query evaluation techniques that soften the transition from safe to unsafe queries. One approach extends the reach of safe plans: for example, safe subplans can be used to speed up processing unsafe queries,³³ functional dependencies on the data, or knowing that some relations are deterministic can be used to find more safe plans,^{11, 29} and safe plans have been described for query languages for streams of events.³⁴

Another approach is to optimize the general-purpose probabilistic inference on the lineage expressions.³⁶

A new direction is taken by a recent project at IBM Almaden²³ that builds a database system where Monte Carlo simulations are pushed deep inside the engine, thus able to evaluate any query safe or unsafe. What is particularly promising about this approach is that through clever query optimization techniques, such as *tuple bundles*, the cost of sampling operations can be drastically reduced. A complementary approach, explored by Olteanu et al.,²⁹ is to rewrite queries into ordered binary decision diagrams (OBDD). They have observed that safe plans lead to linear-sized OBDDs. This raises the possibility that other tractable cases of OBDDs can be inferred, perhaps by analyzing both the query expression and the database statistics.

Materialized Views. The use of materialized views to answer queries is a very powerful tool in data management.²¹ In its most simple formulation, there are a number of materialized views, for example, answers to previous queries, and the query is rewritten in terms of these views, to improve performance.

In the case of probabilistic databases, materialized views have been studied in Ré and Suciu.³³ Because of

the dichotomy of the query complexity, materialized views can have a dramatic impact on query evaluation: a query may be unsafe, hence #P-hard, but after rewriting it in terms of views it may become a safe query, and thus is in PTIME. There is no magic here, we don't avoid the #P-hard problem, we simply take advantage of the fact that the main cost has already been paid when the view was materialized.

The major challenge in using materialized views over probabilistic data is that we need to represent the view's output. We can always compute the lineage of all the tuples in the view, and this provides a complete representation of the view, but it also defeats our purpose, since using these lineage expressions during query evaluation does not simplify the probabilistic inference problem. Instead, we would like to use only the marginal tuple probabilities that have been computed for the materialized view, not their lineage. For example, it may happen that all tuples are independent probabilistic events, and in this case we only need the marginal probabilities; we say in this case the view is fully representable. In general, not all tuples in the view are

independent, but it is always possible to partition the tuples into blocks such that tuples from different blocks are independent, and, moreover, there exists a "best" such partition³³; within a block, the correlations between the tuples remain unspecified. The blocks are described at the schema level, by a specific set of attributes: grouping by those attributes gives the blocks. This is called a *partial representation*, and can be used to evaluate some queries over the views. Note that the problem of finding a good partial representation of the view is done by a static analysis that is orthogonal to the analysis whether the view is safe or unsafe: there are examples for all four combinations of safe/unsafe representable/unrepresentable views.

Facet 3: User Interface

The semantics of query Q on a probabilistic database with possible worlds W is, in theory, quite simple, and is given by the *image probability space* over the set of possible answers, $\{Q(I) \mid I \in W\}$. In practice, it is impossible, and perhaps useless, to return all possible sets of answers. An important problem in probabilistic databases is how to best

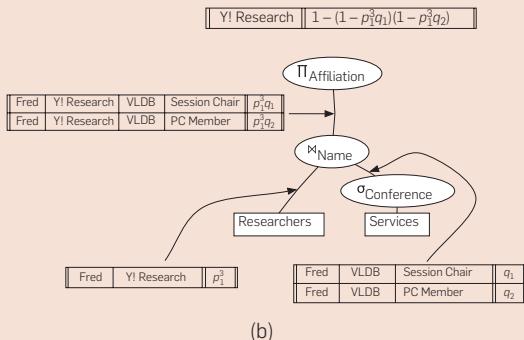
Figure 3: An SQL query on the data in Figure 1(a) returning the affiliations of all researchers who performed some service for VLDB. The query follows the syntax of MayBMS,² where `confidence()` is an aggregate operator returning the output probability. The figure shows an unsafe plan in (b) and a safe plan in (c), and also traces the computation of the output probability of $Y! Research$: it assumes there is a single researcher Fred with that affiliation, and that Fred performed two services for VLDB. The safe plan rewritten in SQL is shown in (d): the aggregate function `prod` is not supported by most relational engines, and needs to be rewritten in terms of `sum`, `logarithms`, and `exponentiation`.

```
SELECT x.Affiliation, confidence()
FROM Researchers x, Services y
WHERE x.Name = y.Name
and y.Conference = 'VLDB'
GROUP BY x.Affiliation
```

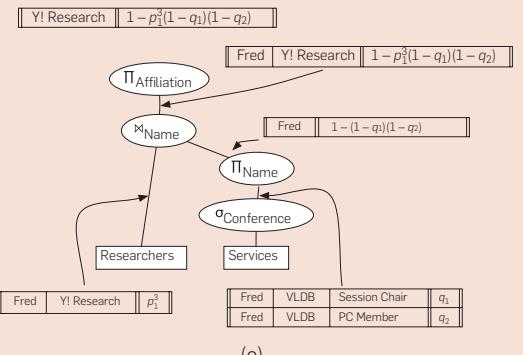
(a)

```
SELECT x.Affiliation, 1-prod(1-x.P*y.P)
FROM Researchers x, (SELECT Name, 1-(1-prod(P))
FROM Services
WHERE Conference = 'VLDB'
GROUP BY Name) y
WHERE x.Name = y.Name
GROUP BY x.Affiliation
```

(d)



(b)



(c)

present the set of possible query answers to the user. To date, two practical approaches have been considered: ranking tuples and aggregation over imprecise values.

Ranking and Top-k Query Answering. In this approach the system returns all possible answer tuples and their probabilities: $P(t_1 \in Q), P(t_2 \in Q), \dots$ noted previously: the correlations between the tuples are thus lost. The emphasis in this approach is to rank these tuples, and restrict them to the top k .

One way to rank tuples is in decreasing order of their output probabilities:³¹ $P(t_1 \in Q) \geq P(t_2 \in Q) \geq \dots$ Often, however, there may be a user-specified order criteria, and then the system needs to combine the user's ranking scores with the output probability.³⁷ A separate question is whether we can use ranking to our advantage to speed up query performance by returning only the k highest ranked tuples: this problem is called *top-k query answering*. One can go a step further and drop the output probabilities altogether: Ré et al.³¹ argue that ranking the output tuples is the only meaningful semantics for the user, and propose focusing the query processor on computing

the ranking, instead of the output probabilities.

The power of top- k query answering in speeding up query processing has been illustrated in a seminal paper by Fagin et al.¹⁶ When applied to probabilistic databases that principle leads to a technique called *multisimulation*.³¹ It assumes that a tuple's probability $P(t \in Q)$ is approximated by an iterative algorithm, like a Monte Carlo simulation: after some number steps n , the probability $P(t \in Q)$ is known to be, with high probability, in an interval $(p - \varepsilon_n, p + \varepsilon_n)$, where ε_n decreases with n . The idea in the multisimulation algorithm is to carefully control how to allocate the simulation steps among all candidate tuples in the query's answer, in order to identify the top- k tuples without wasting iterations on the other tuples. Multisimulation reduces the computation effort roughly by a factor of N/k , where N is the number of all possible answers, and k is the number of top tuples returned to the user.

Aggregates over Imprecise Data. In SQL, aggregates come in two forms: value aggregates, as in *for each company return the sum of the profits in all its units*, and predicate aggregates, as in

return those companies having the sum of profits greater than 1M. Both types of aggregates are needed in probabilistic databases. The first type is interpreted as expected value, and most aggregate functions can be computed easily using the linearity of expectation. For instance, the complexities of computing *sum* and *count* aggregates over a column are the same as the complexities of answering the same query without the aggregate, such as where all possible values of the column are returned along with their probabilities.¹¹ Complexities of computing *min* and *max* are the same as those of computing the underlying queries with the aggregates replaced by projections removing the columns.¹¹ One aggregate whose expected value is more difficult to compute is *average*, which is an important aggregate function for OLAP over imprecise data. One can compute the expected values of *sum* and *count(*)*, but the expected value of *average* is not their ratio. A surprising result was shown by Jayram et al.²⁴ who proved that *average* can be computed efficiently. They give an exact algorithm to compute *average* on a single table in time $O(n \log^2 n)$. They

Figure 4: The dichotomy of conjunctive queries without self-joins on tuple-independent probabilistic databases is captured by Hierarchical Queries.

Hierarchical Queries

In the case of tuple-independent databases (where all tuples are independent) safe queries are precisely the *hierarchical queries*; we define hierarchical queries here.

A conjunctive query is:

$$q(\bar{z}) : - \text{ body}$$

where body consists of a set of subgoals g_1, g_2, \dots, g_k , and \bar{z} are called the head variables. Denote $\text{Vars}(g_i)$ the set of variables occurring in g_i and $\text{Vars}(q) = \bigcup_{i=1,k} \text{Vars}(g_i)$. For each $x \in \text{Vars}(q)$ denote $sg(x) = [g_i \mid x \in \text{Vars}(g_i)]$.

DEFINITION 2.1. Let q be a conjunctive query and \bar{z} its head variables. q is called *hierarchical* if for all $x, y \in \text{Vars}(q) - \bar{z}$, one of the following holds: (a) $sg(x) \subseteq sg(y)$, or (b) $sg(x) \supseteq sg(y)$, or (c) $sg(x) \cap sg(y) = \emptyset$.

A conjunctive query is without self-joins if any two distinct subgoals refer to distinct relation symbols.

THEOREM 2.2 (DICHOTOMY). (Dalvi and Suciu^{11, 12}) Let q be a conjunctive query without self-joins. (1) If q is hierarchical then its data complexity over tuple-independent databases is in PTIME. (2) If q is not hierarchical then its data complexity over tuple-independent databases is #P-hard.

To illustrate the theorem, consider the two queries:

$$\begin{aligned} q_1(z) &: - R(x, z), S(x, y), T(x, z) \\ q_2(z) &: - R(x, z), S(x, y), T(y, z) \end{aligned}$$

In q_1 we have $sg(x) = [R, S, T], sg(y) = [S]$; hence it is hierarchical and can be evaluated in PTIME. In q_2 we have $sg(x) = [R, S], sg(y) = [S, T]$; hence it is nonhierarchical and is #P-hard.

also give efficient algorithms to compute various aggregates when the data is streaming.

The second type of aggregates, those occurring in the HAVING clause of an SQL query, have also been considered.³² In this case, one needs to compute the entire density function of the random variable represented by the aggregate, and this is more difficult than computing the expected value. Similar to safe queries, the density function can sometimes be computed efficiently and exactly, but it is hard in general. Worse, in contrast to safe queries, which can always be efficiently approximated, there exists HAVING queries that do not admit efficient approximations.

A Little History of the (Possible) Worlds

There is a rich literature on probabilistic databases, and we do not aim here to be complete; rather, as in Gombrich's classic *A Little History of the World*, we aim to "catch a glimpse." Early extensions of databases with probabilities date back to Wong⁴⁰ and Cavallo and Pittarelli.⁶ In an influential paper Barbara et al.³ described a probabilistic data model that is quite close to the BID data model, and showed that SQL queries without duplicate elimination or other aggregations can be evaluated efficiently. ProbView²⁷ removed the restriction on queries, but returned confidence intervals instead of probabilities. At about the same time, Fuhr and Roelleke¹⁸ started to use c-tables and lineage for probabilistic databases and showed that every query can be computed this way.

Probabilities in databases have also been studied in the context of "reliability of queries," which quantifies the probability of a query being correct assuming that tuples in the database have some probability of being wrong. Grädel et al.¹⁹ were the first to prove that a simple query can have data complexity that is #P-hard.

Andritsos et al.¹ have applied probabilistic databases to the problem of consistent query answering over inconsistent databases. They observed that the "certain tuples"²¹ to a query over an inconsistent database are precisely the tuples with probability 1 under probabilistic semantics.

The intense interest in probabilistic databases seen today is due to a

number of influential projects: applications to sensor data,^{7, 15} data cleaning,¹ and information extraction,²⁰ the safe plans of Dalvi and Suciu,¹¹ the Trio system⁴ that introduced ULDBs, and the advanced representation systems described in Antova et al.² and Sen and Deshpande.³⁶

Conclusion

Many applications benefit from finding valuable facts in imprecise data, the *diamonds in the dirt*, without having to clean the data first. The goal of probabilistic databases is to make uncertainty a first-class citizen, and to reduce the cost of using such data, or (more likely) to enable applications that were otherwise prohibitively expensive. This article describes some of the recent advances for large-scale query processing on probabilistic databases and their roots in classical data management concepts.

This work was partially supported by NSF Grants IIS-0454425, IIS-0513877, IIS-0713576, and a Gift from Microsoft. An extended version of this work with additional references is available at <http://www.cs.washington.edu/homes/suciu/>. C

References

1. Andritsos, P. and Fuxman, A., Miller, R.J. Clean answers over dirty databases. In *ICDE* (2006).
2. Antova, L., Jansen, T., Koch, C. and Olteanu, D. Fast and simple relational processing of uncertain data. In *ICDE* (2008).
3. Barbara, D., Garcia-Molina, H. and Porter, D. The management of probabilistic data. *IEEE Trans. Knowl. Data Eng.* 4, 5 (1992), 487–502.
4. Benjelloun, O., Sarma, A.D., Halevy, A., Theobald, M. and Widom, J. Databases with uncertainty and lineage. *VLDBJ* 17, 2 (2008), 243–264.
5. Burdick, D., Deshpande, P., Jayram, T.S., Ramakrishnan, R. and Vaithyanathan, S. Efficient allocation algorithms for OLAP over imprecise data. In *VLDB* (2006), 391–402.
6. Cavallo, R. and Pittarelli, M. The theory of probabilistic databases. In *Proceedings of VLDB* (1987), 71–81.
7. Cheng, R., Kalashnikov, D. and Prabhakar, S. Evaluating probabilistic queries over imprecise data. In *SIGMOD* (2003), 551–562.
8. Codd, E.F. Relational completeness of data base sublanguages. In *Database Systems* (1972), Prentice-Hall, 65–98.
9. Cowell, R., Dawid, P., Lauritzen, S. and Spiegelhalter, D., eds. *Probabilistic Networks and Expert Systems* (1999), Springer.
10. Dalvi, N. and Suciu, D. The dichotomy of conjunctive queries on probabilistic structures. In *PODS* (2007), 293–302.
11. Dalvi, N. and Suciu, D. Efficient query evaluation on probabilistic databases. *VLDB J.* 16, 4 (2007), 523–544.
12. Dalvi, N. and Suciu, D. Management of probabilistic data: Foundations and challenges. In *PODS* (Beijing, China, 2007) 1–12 (invited talk).
13. Darwiche, A. A differential approach to inference in bayesian networks. *J. ACM* 50, 3 (2003), 280–305.
14. DeRose, P., Shen, W., Chen, F., Lee, Y., Burdick, D., Doan, A. and Ramakrishnan, R. Dblife: A community information management platform for the database research community. In *CIDR* (2007), 169–172.
15. Deshpande, A., Guestrin, C., Madden, S., Hellerstein, J.M. and Hong, W. Model-driven data acquisition in sensor networks. In *VLDB* (2004), 588–599.
16. Fagin, R., Lotem, A. and Naor, M. Optimal aggregation algorithms for middleware. In *Proceedings of the 20th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems* (2001), ACM Press, 102–113.
17. Friedman, N., Getoor, L., Koller, D. and Pfeffer A. Learning probabilistic relational models. In *IJCAI* (1999), 1300–1309.
18. Fuhr, N. and Roelleke, T. A probabilistic relational algebra for the integration of information retrieval and database systems. *ACM Trans. Inf. Syst.* 15, 1 (1997), 32–66.
19. Grädel, E., Gurevich, Y. and Hirsch, C. The complexity of query reliability. In *PODS* (1998), 227–234.
20. Gupta, R. and Sarawagi, S. Creating probabilistic databases from information extraction models. In *VLDB* (2006), 965–976.
21. Halevy, A. Answering queries using views: A survey. *VLDB J.* 10, 4 (2001), 270–294.
22. Imielinski, T. and Lipski, W. Incomplete information in relational databases. *J. ACM* 31 (Oct. 1984), 761–791.
23. Jampani, R., Xu, F., Wu, M., Perez, L., Jermaine, C. and Haas, P. MCDB: A Monte Carlo approach to managing uncertain data. In *SIGMOD* (2008), 687–700.
24. Jayram, T., Kale, S. and Vee, E. Efficient aggregation algorithms for probabilistic data. In *SODA* (2007).
25. Kanagal, B. and Deshpande, A. Online filtering, smoothing and probabilistic modeling of streaming data. In *ICDE* (2008), 1160–1169.
26. Lafferty, J., McCallum, A. and Pereira, F. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML* (2001).
27. Lakshmanan, L., Leone, N., Ross, R. and Subrahmanian, V. Proview: A flexible probabilistic database system. *ACM Trans. Database Syst.* 22, 3 (1997).
28. Nierman, A. and Jagadish, H. ProTDB: Probabilistic data in XML. In *VLDB* (2002), 646–657.
29. Olteanu, D., Huang, J. and Koch, C. SPROUT: Lazy vs. eager query plans for tuple independent probabilistic databases. In *ICDE* (2009).
30. Rastogi, V., Suciu, D. and Hong, S. The boundary between privacy and utility in data publishing. In *VLDB* (2007).
31. Ré, C., Dalvi, N. and Suciu, D. Efficient Top-k query evaluation on probabilistic data. In *ICDE* (2007).
32. Ré, C., Suciu, D. Efficient evaluation of having queries on a probabilistic database. In *Proceedings of DBPL* (2007).
33. Ré, C. and Suciu, D. Materialized views in probabilistic databases for information exchange and query optimization. In *Proceedings of VLDB* (2007).
34. Ré, C., Letchner, J., Balazinska, M. and Suciu, D. Event queries on correlated probabilistic streams. In *SIGMOD* (Vancouver, Canada, 2008).
35. Roth, D. On the hardness of approximate reasoning. *Artif. Intell.* 82, 1–2 (1996), 273–302.
36. Sen, P. and Deshpande, A. Representing and querying correlated tuples in probabilistic databases. In *ICDE*, 2007.
37. Soliman, M.A., Ilyas, I.F. and Chang, K.C.-C. Probabilistic top- and ranking-aggregate queries. *ACM Trans. Database Syst.* 33, 3 (2008).
38. Vardi, M.Y. The complexity of relational query languages. In *Proceedings of 14th ACM SIGACT Symposium on the Theory of Computing* (San Francisco, California, 1982), 137–146.
39. Verma, T. and Pearl, J. Causal networks: Semantics and expressiveness. *Uncertainty Artif. Intell.* 4 (1990), 69–76.
40. Wong, E. A statistical approach to incomplete information in database systems. *ACM Trans. Database Syst.* 7, 3 (1982), 470–488.

Nilesh Dalvi (ndalvi@yahoo-inc.com)
Yahoo!Research, Santa Clara, CA.

Christopher Ré (chrisre@cs.washington.edu)
University of Washington, Seattle, WA.

Dan Suciu (suciu@cs.washington.edu)
University of Washington, Seattle, WA.

research highlights

P. 96

Technical Perspective The Ultimate Pilot Program

By Stuart Russell
and Lawrence Saul

P. 97

Apprenticeship Learning for Helicopter Control

By Adam Coates, Pieter Abbeel, and Andrew Y. Ng

P. 106

Technical Perspective A Compiler's Story

By Greg Morrisett

P. 107

Formal Verification of a Realistic Compiler

By Xavier Leroy

Technical Perspective

The Ultimate Pilot Program

By Stuart Russell and Lawrence Saul

IN ONE SCENE from *The Matrix*, two leaders of the human resistance are trapped on the roof of a skyscraper. The only means of escape is by helicopter, which neither can operate. The humans quickly call up a “pilot program” for helicopter flight, absorb the knowledge instantly via a brain-computer interface, and take off in the nick of time.

The following paper by Coates, Abbeel, and Ng describes an equally remarkable feat: learning to fly helicopter aerobatics of superhuman quality by watching a few minutes of a human expert performance. Before you read the paper, we suggest watching the videos at <http://heli.stanford.edu/>.

The authors provide careful descriptions of the problem and of the technical innovations required for its solution. The paper’s importance lies not only in these innovations, but also in the way it illustrates the flavor of modern artificial intelligence research. AI has grown to encompass, in a seamless way, techniques from areas such as statistical learning, dynamical systems, and control theory, and has reintegrated with areas that many thought had gone their own way, such as robotics, vision, and natural language understanding. The key to reunification has been the emergence of effective techniques for probabilistic reasoning and machine learning. The authors illustrate this trend perfectly, solving a problem in robotics that had resisted traditional control theory techniques for many years.

Learning to fly a helicopter means learning a policy—a mapping from states to control actions. What form should the mapping take and what information should be supplied to the learning system? Some early work adopted the idea of observing expert performance to learn to fly a small plane,¹ using *supervised* learning methods and representing policies as decision trees. In this approach, each expert action is a positive example of the function to be learned, while each ac-

tion not taken is a negative example. Unfortunately, the resulting policies fail miserably when any perturbation puts the aircraft into a state not seen during training. Perhaps this is not surprising, because the policy has no idea how the vehicle works or what the pilot is attempting.

In contrast, the authors formulate the problem as a Markov decision process (MDP), where the *transition model* specifies how the vehicle works, the *reward function* specifies what the pilot is trying to do, and the *optimal policy* maximizes the expected sum of rewards over the entire trajectory. Initially, of course, the transition model and reward function are unknown, so the learning system cannot compute the optimal policy without first obtaining more information. In the well-established setting of *reinforcement learning*, the learning system acts in the world and observes outcomes and rewards. For many problems, learning a model and a reward function requires fewer experiences than trying to learn a policy directly—and experiences are always in short supply in robot learning.

Pure *tabula rasa* reinforcement learning is not applicable to helicopter aerobatics, however, for two reasons: First, in the early stages of learning there would be far too many crashes; second, the reward function is not known even to the experimenters, so a reward signal cannot easily be provided to the learning system. The *apprenticeship learning* setting adopted by the authors avoids both problems by learning from expert behaviors.

By observing the helicopter’s trajectory while the expert is flying, the learning system can acquire a transition model that is reasonably accurate in the regions of state space that are likely to be visited during these maneuvers. The role of *prior knowledge* is crucial here; while the model parameters are learned, the model structure is determined in advance from general knowledge of helicopter dynamics.

The task of learning the reward

function from expert behavior is called “inverse reinforcement learning.” Introduced in AI in the late 1990s, this actually has a long history in economics.² For helicopter aerobatics, the reward function specifies what the desirable trajectories are, such that following them yields high reward, and how deviations should be penalized. This information is implicit in the expert’s behavior and its variability. To account for this variability, the authors develop a probabilistic generative model for trajectories, borrowing methods from speech recognition and biological sequence alignment to handle variations in timing. After learning from several expert performances, the reward function actually defines a much better trajectory than the expert could demonstrate, and the autonomous helicopter eventually outperforms its human teacher.

The authors’ success in this difficult task reflects fundamental progress in our field. While achieving comparable success on other difficult robotic tasks is not yet a routine application of off-the-shelf methods, the technology of apprenticeship learning provides a plausible template for progress. ■

References

1. Sammut, C., Hurst, S., Kedzier, D. and Michie, D. Learning to fly. In *Proceedings of the Intern. Conf. on Machine Learning* (1992).
2. Sargent, T.J. Estimation of dynamic labor demand schedules under rational expectations. *J. Political Economy* 86 (1978), 1009–1044.

Stuart Russell is a professor of CS, chair of the Department of Electrical Engineering and Computer Sciences at the University of California, Berkeley, and co-chair of Communications’ Research Highlights Board.

Lawrence Saul is an associate professor in the Department of Computer Science and Engineering at the University of California, San Diego, and a member of Communications’ Research Highlights Board.

Apprenticeship Learning for Helicopter Control

By Adam Coates, Pieter Abbeel, and Andrew Y. Ng

Abstract

Autonomous helicopter flight is widely regarded to be a highly challenging control problem. As helicopters are highly unstable and exhibit complicated dynamical behavior, it is particularly difficult to design controllers that achieve high performance over a broad flight regime.

While these aircraft are notoriously difficult to control, there are expert human pilots who are nonetheless capable of demonstrating a wide variety of maneuvers, including aerobatic maneuvers at the edge of the helicopter's performance envelope. In this paper, we present algorithms for modeling and control that leverage these demonstrations to build high-performance control systems for autonomous helicopters. More specifically, we detail our experiences with the Stanford Autonomous Helicopter, which is now capable of extreme aerobatic flight meeting or exceeding the performance of our own expert pilot.

1. INTRODUCTION

Autonomous helicopter flight represents a challenging control problem with high-dimensional, asymmetric, noisy, nonlinear, nonminimum phase dynamics. Helicopters are widely regarded to be significantly harder to control than fixed-wing aircraft. (See, e.g., Leishman,¹⁸ Seddon.³¹) At the same time, helicopters provide unique capabilities, such as in-place hover and low-speed flight, important for many applications. The control of autonomous helicopters thus provides a challenging and important test bed for learning and control algorithms.

There is a considerable body of research concerning control of autonomous (RC) helicopters in the typical "upright flight regime." This has allowed autonomous helicopters to reliably perform many practical maneuvers, such as sustained hover, low-speed horizontal flight, and autonomous landing.^{9,16,17,24,28,30}

In contrast, autonomous flight achievements in other flight regimes have been limited. Gavrilets et al.¹⁴ performed some of the first autonomous aerobatic maneuvers: a stall-turn, a split-S, and an axial roll. Ng et al.²³ achieved sustained autonomous inverted hover. While these results significantly expanded the potential capabilities of autonomous helicopters, it has remained difficult to design control systems capable of performing arbitrary aerobatic maneuvers at a performance level comparable to human experts.

In this paper, we describe our line of autonomous helicopter research. Our work covers a broad approach to autonomous helicopter control based on "apprenticeship learning" that achieves expert-level performance on a vast array of maneuvers, including extreme aerobatics and autonomous autorotation landings.^{1,2,12,23} (Refer footnote a.)

In apprenticeship learning, we assume that an expert is available who is capable of performing the desired maneuvers. We then leverage these demonstrations to learn all of the necessary components for our control system. In particular, the demonstrations allow us to learn a model of the helicopter dynamics, as well as appropriate choices of target trajectories and reward parameters for input into a reinforcement learning or optimal control algorithm.

The remainder of this paper is organized as follows: Section 2 briefly overviews related work in the robotics literature that is similar in spirit to our approach. Section 3 describes our basic modeling approach, where we develop a model of the helicopter dynamics from data collected under human control, and subsequently improve this model using data from autonomous flights. Section 4 presents an apprenticeship-based trajectory learning algorithm that learns idealized trajectories of the maneuvers we wish to fly. This algorithm also provides a mechanism for improving our model of the helicopter dynamics along the desired trajectory. Section 5 describes our control algorithm, which is based on differential dynamic programming (DDP).¹⁵ Section 6 describes our helicopter platform and presents our experimental results.

2. RELATED WORK

Although no prior works span our entire setting of apprenticeship learning for control, there are separate pieces of work that relate to various components of our approach.

Atkeson and Schaal,⁸ for instance, use multiple demonstrations to learn a model for a robot arm, and then find an optimal controller in their simulator, initializing their optimal control algorithm with one of the demonstrations.

The work of Calinon et al.¹¹ considered learning trajectories and constraints from demonstrations for robotic tasks. There, however, they do not consider the system's dynamics or provide a clear mechanism for the inclusion of prior knowledge, which will be a key component of our approach as detailed in Section 4. Our formulation will present a principled, joint optimization which takes into account the multiple demonstrations, as well as the (complex) system dynamics.

Among others, An et al.⁶ and Abbeel et al.⁵ have exploited the idea of trajectory-specific model learning for control.

^a Autorotation is an emergency maneuver that allows a trained pilot to descend and land the helicopter without engine power.

A previous version of this paper, entitled "Learning for Control from Multiple Demonstrations" was published in *Proceedings of the 26th International Conference of Machine Learning*, (ICML 2008), 144–151.

In contrast to our setting, though, their algorithms do not coherently integrate data from multiple (suboptimal) demonstrations by experts. We will nonetheless use similar ideas in our trajectory learning algorithm.

Our work also has strong connections with recent work on inverse reinforcement learning, which extracts a reward function from expert demonstrations. See, e.g., Abbeel,⁴ Neu,²² Ng, Ramachandran, Ratliff,^{25–27} Syed.³² We will describe a methodology roughly corresponding to the inverse RL algorithm of Abbeel⁴ to tune reward weights in Section 5.2.

3. MODELING

The helicopter state s comprises its position (x, y, z) , orientation (expressed as a unit quaternion q), velocity $(\dot{x}, \dot{y}, \dot{z})$, and angular velocity $(\omega_x, \omega_y, \omega_z)$. The pitch angle of a blade is changed by rotating it around its long axis changing the amount of thrust the blade generates. The helicopter is controlled via a four-dimensional action space:

1. u_1 and u_2 : The lateral (left-right) and longitudinal (front-back) cyclic pitch controls cause the helicopter to roll left or right, and pitch forward or backward, respectively.
2. u_3 : The tail rotor pitch control changes tail rotor thrust, controlling the rotation of the helicopter about its vertical axis.
3. u_4 : The main rotor collective pitch control changes the pitch angle of the main rotor's blades, by rotating the blades around an axis that runs along the length of the blade. The resulting amount of upward thrust (generally) increases with this pitch angle; thus this control affects the main rotor's thrust.

By using the cyclic pitch and tail rotor controls, the pilot can rotate the helicopter into any orientation. This allows the pilot to direct the thrust of the main rotor in any particular direction (and thus fly in any particular direction) by rotating the helicopter appropriately.

Following our approach from Abbeel,³ we learn a model from flight data that predicts accelerations as a function of the current state and inputs. Accelerations are then integrated to obtain the state changes over time. To take advantage of symmetry of the helicopter, we predict linear and angular accelerations in a “body-coordinate frame” (a coordinate frame attached to the helicopter). In this body-coordinate frame, the x -axis always points forward, the y -axis always points to the right, and z -axis always points down with respect to the helicopter.

In particular, we use the following model:

$$\begin{aligned}\ddot{x}^b &= A_x \dot{x}^b + g_x^b + w_x, \\ \ddot{y}^b &= A_y \dot{y}^b + g_y^b + D_0 + w_y, \\ \ddot{z}^b &= A_z \dot{z}^b + g_z^b + C_4 u_4 + D_4 + w_z, \\ \dot{\omega}_x^b &= B_x \omega_x^b + C_1 u_1 + D_1 + w_{\omega_x}, \\ \dot{\omega}_y^b &= B_y \omega_y^b + C_2 u_2 + D_2 + w_{\omega_y}, \\ \dot{\omega}_z^b &= B_z \omega_z^b + C_3 u_3 + D_3 + w_{\omega_z}.\end{aligned}$$

By our convention, the superscripts b indicate that we are using body coordinates. We note our model explicitly encodes the dependence on the gravity vector (g_x^b, g_y^b, g_z^b) and has a sparse dependence of the accelerations on the current velocities, angular rates, and inputs. The terms $w_x, w_y, w_z, w_{\omega_x}, w_{\omega_y}$, and w_{ω_z} are zero mean Gaussian random variables, which represent the perturbation of the accelerations due to noise (or unmodeled effects).

To learn the coefficients, we record data while the helicopter is being flown by our expert pilot. We typically ask our pilot to fly the helicopter through the flight regimes we would like to model. For instance, to build a model for hovering, the pilot places the helicopter in a stable hover and sweeps the control sticks back and forth at varying frequencies to demonstrate the response of the helicopter to different inputs while hovering. Once we have collected this data, the coefficients (e.g., A_x, B_x, C_1 , etc.) are estimated using linear regression.

When we want to perform a new maneuver, we can collect data from the flight regimes specific to this maneuver and build a new model. For aerobatic maneuvers, this involves having our pilot repeatedly demonstrate the desired maneuver.

It turns out that, in practice, these models generalize reasonably well and can be used as a “crude” starting point for performing aerobatic maneuvers. In previous work,² we demonstrated that models of the above form are sufficient for performing several maneuvers including “funnels” (fast sideways flight in a circle) and in-place flips and rolls. With a “crude” model trained from demonstrations of these maneuvers, we can attempt the maneuver autonomously. If the helicopter does not complete the maneuver successfully, the model can be re-estimated, incorporating the data obtained during the failed trial. This new model more accurately captures the dynamics in the flight regimes actually encountered during the autonomous flight and hence can be used to achieve improved performance during subsequent attempts.

The observation that we can leverage pilot demonstrations to safely obtain “reasonable” models of the helicopter dynamics is the key to our approach. While these models may not be perfect at first, we can often obtain a good approximation to the true dynamics provided we attempt to model only a small portion of the flight envelope. This model can then, optionally, be improved by incorporating new data obtained from autonomous flights. Our trajectory learning algorithm (Section 4) exploits this same observation to achieve expert-level performance on an even broader range of maneuvers.

4. TRAJECTORY LEARNING

Once we are equipped with a (rudimentary) model of the helicopter dynamics, we need to specify the desired trajectory to be flown. Specifying the trajectory by hand, while tedious, can yield reasonable results. Indeed, much of our own previous work used hand-coded target trajectories.² Unfortunately these trajectories usually do not obey the system dynamics—that is, the hand-specified trajectory is infeasible, and cannot actually be flown in reality. This results in a somewhat more difficult control problem since

the control algorithm must determine an appropriate trade-off between the errors it must inevitably make. As well, it complicates our modeling process because we do not know, *a priori*, the trajectory that the controller will attempt to fly, and hence cannot focus our data collection in that region of state space.

One solution to these problems is to leverage expert demonstrations. By using a trajectory acquired from a demonstration aboard the real helicopter as the target trajectory we are guaranteed that our target is a feasible trajectory. Moreover, our data collection will already be focused on the proper flight regime, provided that our expert demonstrations cover roughly the same parts of state space each time. Thus, we expect that our model of the dynamics along the demonstrated trajectory will be reasonably accurate. This approach has been used successfully to perform autonomous autorotation landings with our helicopter.¹

While the autorotation maneuver can be demonstrated relatively consistently by a skilled pilot,^b it may be difficult or impossible to obtain a perfect demonstration that is suitable for use as a target trajectory when the maneuver does not include a steady-state regime, or involves complicated adjustments over long periods of time. For example, when our expert pilot attempts to demonstrate an in-place flip, the helicopter position often drifts away from its starting point unintentionally. Thus, when using this demonstration as our desired trajectory, the helicopter will repeat the pilot's errors. However, repeated expert demonstrations are often suboptimal in different ways, suggesting that a large number of demonstrations could implicitly encode the ideal trajectory that the (suboptimal) expert is trying to demonstrate.

In Coates,¹² we proposed an algorithm that approximately extracts this implicitly encoded optimal demonstration from multiple suboptimal expert demonstrations. This algorithm also allows us to build an improved, time-varying model of the dynamics along the resulting trajectory suitable for high-performance control. In doing so, the algorithm allows the helicopter to not only mimic the behavior of the expert but even perform significantly better.

Properly extracting the underlying ideal trajectory from a set of suboptimal trajectories requires a significantly more sophisticated approach than merely averaging the states observed at each time step. A simple arithmetic average of the states would result in a trajectory that does not obey the constraints of the dynamics model. Also, in practice, each of the demonstrations will occur at different rates so that attempting to combine states from the same time step in each trajectory will not work properly.

Following Coates,¹² we propose a generative model that describes the expert demonstrations as noisy observations of the unobserved, intended target trajectory, where each demonstration is possibly warped along the time axis. We use an expectation–maximization (EM) algorithm to both infer the unobserved, intended target trajectory and a time-alignment

of all the demonstrations. The time-aligned demonstrations provide the appropriate data to learn good local models in the vicinity of the trajectory—such trajectory-specific local models tend to greatly improve control performance.

4.1. Basic generative model

From our expert pilot we obtain M demonstration trajectories of length N^k , for $k = 0..M - 1$. Each trajectory is a sequence of states, s_j^k , and control inputs, u_j^k , composed into a single state vector:

$$y_j^k = \begin{bmatrix} s_j^k \\ u_j^k \end{bmatrix} \text{ for } j = 0..N^k - 1, k = 0..M - 1.$$

Our goal is to estimate a “hidden” target trajectory of length H , denoted similarly:

$$z_t = \begin{bmatrix} s_t^* \\ u_t^* \end{bmatrix} \text{ for } t = 0..H.$$

We use the following notation: $y = \{y_j^k \mid j = 0..N^k - 1, k = 0..M - 1\}$, $\mathbf{z} = \{z_t \mid t = 0..H\}$, and similarly for other indexed variables.

The generative model for the ideal trajectory is given by an initial state distribution $z_0 \sim \mathcal{N}(\mu_0, \Sigma_0)$ and an approximate model of the dynamics

$$z_{t+1} = f(z_t) + w_t^{(z)}, \quad w_t^{(z)} \sim \mathcal{N}(0, \Sigma^{(z)}). \quad (1)$$

The dynamics model does not need to be particularly accurate. In fact, in our experiments, this model is of the form described in Section 3, trained on a large corpus of data that is not even specific to the trajectory we want to fly.^c In our experiments (Section 6) we provide some concrete examples showing how accurately the generic model captures the true dynamics for our helicopter.

Our generative model represents each demonstration as a set of independent “observations” of the hidden, ideal trajectory \mathbf{z} . Specifically, our model assumes

$$y_j^k = z_{\tau_j^k} + w_j^{(y)}, \quad w_j^{(y)} \sim \mathcal{N}(0, \Sigma^{(y)}). \quad (2)$$

Here τ_j^k is the time index in the hidden trajectory to which the observation y_j^k is mapped. The noise term in the observation equation captures both inaccuracies in estimating the observed trajectories from sensor data, as well as errors in the maneuver that are the result of the human pilot's imperfect demonstration.^d

^b The autorotation maneuver consists of a steady-state “glide” followed by a short (several second) “flare” before landing. Though the maneuver is not easy to learn, these components tend not to vary much from one demonstration to the next.

^c The state transition model also predicts the controls as a function of the previous state and controls. In our experiments we predict u_{t+1}^* as u_t^* plus Gaussian noise.

^d Even though our observations, \mathbf{y} , are correlated over time with each other due to the dynamics governing the observed trajectory, our model assumes that the observations y_j^k are independent for all $j = 0..N^k - 1$ and $k = 0..M - 1$.

The time indices τ_j^k are unobserved, and our model assumes the following distribution with parameters d_i^k :

$$\mathbb{P}(\tau_{j+1}^k | \tau_j^k) = \begin{cases} d_1^k & \text{if } \tau_{j+1}^k - \tau_j^k = 1, \\ d_2^k & \text{if } \tau_{j+1}^k - \tau_j^k = 2, \\ d_3^k & \text{if } \tau_{j+1}^k - \tau_j^k = 3, \\ 0 & \text{otherwise,} \end{cases} \quad (3)$$

$$\tau_0^k \equiv 0. \quad (4)$$

To accommodate small, gradual shifts in time between the hidden and observed trajectories, our model assumes the observed trajectories are subsampled versions of the hidden trajectory. We found that having a hidden trajectory length equal to twice the average length of the demonstrations, i.e., $H = 2(\frac{1}{M} \sum_{k=0}^{M-1} N^k)$, gives sufficient resolution.

Figure 1 depicts the graphical model corresponding to our basic generative model. Note that each observation y_j^k depends on the hidden trajectory's state at time τ_j^k , which means that for τ_j^k unobserved, y_j^k depends on all states in the hidden trajectory with which it could potentially be associated.

4.2. Extensions to the generative model

We have assumed, thus far, that the expert demonstrations are misaligned copies of the ideal trajectory merely corrupted by Gaussian noise. Listgarten et al. have used this same basic generative model (for the case where $f(\cdot)$ is the identity function) to align speech signals and biological data.^{19,20} In our application to autonomous helicopter flight, we can augment the basic model described above to account for other sources of error that are important for modeling and control.

Learning Local Model Parameters: We can substantially improve our modeling accuracy by using a time-varying model $f_t(\cdot)$ that is specific to the vicinity of the intended trajectory at each time t .

We express f_t as our “crude” model (from Section 3), f , augmented with a bias term, ^e β_t^* :

$$z_{t+1} = f_t(z_t) + w_t^{(z)} \equiv f(z_t) + \beta_t^* + w_t^{(z)}.$$

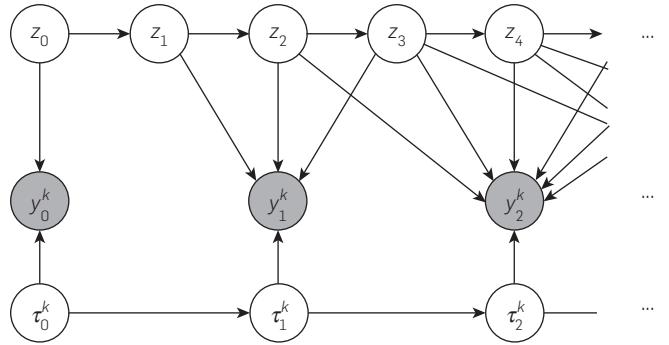
To regularize our model, we assume that β_t^* changes only slowly over time. Specifically $\beta_{t+1}^* \sim \mathcal{N}(\beta_t^*, \Sigma^{(\beta)})$.

We incorporate the bias into our observation model by computing the observed bias $\beta_j^k = y_j^k - f(y_{j-1}^k)$ for each of the observed state transitions, and modeling this as a direct observation of the “true” model bias corrupted by Gaussian noise.

The result of this modification is that the ideal trajectory must not only look similar to the demonstration trajectories, but it must also obey a dynamics model which

^e Our generative model can incorporate richer local models. We discuss our choice of merely using biases in Coates.¹² We also show there how to estimate richer models post hoc using the output of our trajectory learning algorithm.

Figure 1: Graphical model representing our trajectory assumptions. (Shaded nodes are observed.)



includes those modeling errors consistently observed in the demonstrations.

Factoring Out Demonstration Drift: It is often difficult, even for an expert pilot, during aerobatic maneuvers to keep the helicopter centered around a fixed position. The recorded position trajectory will often drift around unintentionally. Since these position errors are highly correlated, they are not explained well by the Gaussian noise term in the observation model. The basic dynamics model is easily augmented with “drift” terms to model these errors, allowing us to infer the drift included in each demonstration and remove it from the final result (see Coates¹² for details).

Incorporating Prior Knowledge: Even though it might be hard to specify the complete ideal trajectory in state space, we might still have prior knowledge about the trajectory. For example, for the case of a helicopter performing an in-place flip, our expert pilot can tell us that the helicopter should stay at a fixed position while it is flipping. We show in Coates¹² that these bits of knowledge can be incorporated into our model as additional noisy observations of the hidden states, where the variance of the noise expresses our confidence in the accuracy of the expert’s advice. In the case of the flip, the variance expresses our knowledge that it is, in fact, impossible to flip perfectly in place and that the actual position of the helicopter may vary slightly from the position given by the expert.

4.3. Trajectory learning algorithm

Our learning algorithm will automatically find the time-alignment indices τ , the time-index transition probabilities \mathbf{d} , and the covariance matrices $\Sigma^{(\cdot)}$ by (approximately) maximizing the joint likelihood of the observed trajectories \mathbf{y} and the observed prior knowledge about the ideal trajectory, ρ , while marginalizing out over the unobserved, intended trajectory \mathbf{z} . Concretely, our algorithm (approximately) solves

$$\max_{\tau, \Sigma^{(\cdot)}, \mathbf{d}} \log \mathbb{P}(\mathbf{y}, \rho, \tau; \Sigma^{(\cdot)}, \mathbf{d}). \quad (5)$$

Then, once our algorithm has found $\tau, \mathbf{d}, \Sigma^{(\cdot)}$, it finds the most likely hidden trajectory, namely the trajectory \mathbf{z} that maximizes the joint likelihood of the observed trajectories \mathbf{y} and the observed prior knowledge about the ideal trajectory for the learned parameters $\tau, \mathbf{d}, \Sigma^{(\cdot)}$. The joint optimization in

Equation 5 is difficult because (as can be seen in Figure 1) the lack of knowledge of the time-alignment index variables τ introduces a very large set of dependencies between all the variables. However, when τ is known, the optimization problem in Equation 5 greatly simplifies thanks to context specific independencies.¹⁰ For instance, knowledge that $\tau_1^k = 3$ tells us that y_1^k depends only on z_3 . Thus, when all of the τ are fixed, we obtain a simplified model such as the one shown in Figure 2. In this model we can directly estimate the multinomial parameters \mathbf{d} in closed form; and we have a standard HMM parameter learning problem for the covariances $\Sigma^{(*)}$, which can be solved using the EM algorithm¹³—often referred to as Baum–Welch in the context of HMMs. Concretely, for our setting, the EM algorithm’s E-step computes the pairwise marginals over sequential hidden state variables by running a (extended) Kalman smoother; the M-step then uses these marginals to update the covariances $\Sigma^{(*)}$.

To also optimize over the time-indexing variables τ , we propose an alternating optimization procedure. For fixed $\Sigma^{(*)}$ and \mathbf{d} , and for fixed \mathbf{z} , we can find the optimal time-indexing variables τ using dynamic programming over the time-index assignments for each demonstration independently. The dynamic programming algorithm to find τ is known in the speech recognition literature as dynamic time warping²⁹ and in the biological sequence alignment literature as the Needleman–Wunsch algorithm.²¹ The fixed \mathbf{z} we use is the one that maximizes the likelihood of the observations for the current setting of parameters $\tau, \mathbf{d}, \Sigma^{(*)}$.^f

In practice, rather than alternating between complete optimizations over $\Sigma^{(*)}$, \mathbf{d} and τ , we only partially optimize over $\Sigma^{(*)}$, running only one iteration of the EM algorithm.

Complete details of the algorithm are provided in Coates.¹²

5. CONTROLLER DESIGN

Using the methods of Sections 3 and 4, we can obtain a good target trajectory and a high-accuracy dynamics model for this trajectory using pilot demonstrations. It remains to develop an adequate feedback controller that will allow the helicopter to fly this trajectory in reality. Our solution is based on the DDP algorithm, which we have used in previous work.^{1,2}

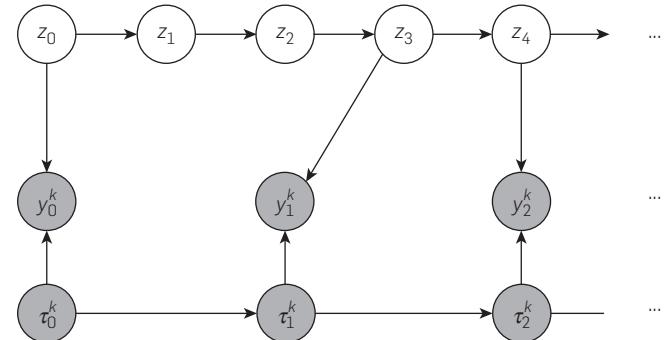
5.1. Reinforcement learning formalism and DDP

A reinforcement learning problem (or optimal control problem) can be described by a quintuple $(S, \mathcal{A}, \mathcal{T}, H, s_0, R)$, which is also referred to as a Markov decision process (MDP). Here S is the set of states; \mathcal{A} is the set of actions or inputs; \mathcal{T} is the dynamics model, which is a set of probability distributions $\{P_{su}^t\}$ ($P_{su}^t(s' | s, u)$ is the probability of being in state s' at time $t+1$ given the state and action at time t are s and u); H is the horizon or number of time steps of interest; $s_0 \in S$ is the initial state; $R: S \times \mathcal{A} \rightarrow \mathbb{R}$ is the reward function.

A policy $\pi = (\mu_0, \mu_1, \dots, \mu_H)$ is a tuple of mappings from states S to actions \mathcal{A} , one mapping for each time $t = 0, \dots, H$.

^f Fixing \mathbf{z} means the dynamic time warping step only approximately optimizes the original objective. Unfortunately, without fixing \mathbf{z} , the independencies required to obtain an efficient dynamic programming algorithm do not hold. In practice we find our approximation works very well.

Figure 2: Example of graphical model when τ is known. (Shaded nodes are observed.)



The expected sum of rewards when acting according to a policy π is given by: $E[\sum_{t=0}^H R(s_t, u_t) | \pi]$. The optimal policy π^* for an MDP $(S, \mathcal{A}, \mathcal{T}, H, s_0, R)$ is the policy that maximizes the expected sum of rewards. In particular, the optimal policy is given by

$$\pi^* = \arg \max_{\pi} E \left[\sum_{t=0}^H R(s_t, u_t) | \pi \right].$$

The linear quadratic regulator (LQR) control problem is a special class of MDP, for which the optimal policy can be computed efficiently. In LQR the set of states $S = \mathbb{R}^n$, the set of actions/inputs $\mathcal{A} = \mathbb{R}^p$, the dynamics model is given by

$$s_{t+1} = A_t s_t + B_t u_t + w_t,$$

where for all $t=0, \dots, H$ we have that $A_t \in \mathbb{R}^{n \times n}$, $B_t \in \mathbb{R}^{n \times p}$ and w_t is a mean zero random variable (with finite variance). The reward for being in state s_t and taking action/input u_t is given by

$$-s_t^\top Q_t s_t - u_t^\top R_t u_t.$$

Here Q_t, R_t are positive semidefinite matrices which parameterize the reward function. It is well known that the optimal policy for the LQR control problem is a time-varying linear feedback controller, which can be efficiently computed using dynamic programming. (See, e.g., Anderson⁷ for details on linear quadratic methods.)

The linear quadratic methods, which in their standard form as given above drive the state to zero, are easily extended to the task of tracking the desired trajectory s_0^*, \dots, s_H^* learned in Section 4. The standard formulation (which we use) expresses the dynamics and reward function as a function of the error state $e_t = s_t - s_t^*$ rather than the actual state s_t . (See, e.g., Anderson.⁷)

DDP approximately solves general continuous state-space MDP’s by iteratively approximating them by LQR problems. In particular, DDP solves an optimal control problem by iterating the following steps:

1. Around the trajectory obtained from running the current policy, compute: (i) a linear approximation to the

- (nonlinear) error state dynamics and (ii) a quadratic approximation to the reward function.
2. Compute the optimal policy for the LQR problem obtained in Step 2 and set the current policy equal to the optimal policy for the LQR problem.
 3. Simulate a trial starting from, s_0 , under the current policy and store the resulting trajectory.

In our experiments, we have a quadratic reward function, thus the only approximation made in the algorithm is the linearization of the dynamics. To bootstrap the process (i.e., to obtain an initial trajectory), we linearize around the target trajectory in the first iteration.

The result of DDP is a sequence of linear feedback controllers that are executed in order. Since these controllers were computed under the assumption of linear dynamics, they will generally fail if executed from a state that is far from the linearization point. For aerobatic maneuvers that involve large changes in orientation, it is often difficult to remain sufficiently close to the linearization point throughout the maneuver. Our system, thus, uses DDP in a “receding horizon” fashion. Specifically, we rerun DDP *online*, beginning from the current state of the helicopter, over a horizon that extends 2 s into the future.^g The resulting feedback controller obtained from this process is always linearized around the current state and, thus, allows the control system to continue flying even when it ventures briefly away from the intended trajectory.

5.2. Learning reward function parameters

Our quadratic reward is a function of 21 features (which are functions of the state and controls), consisting of the squared error state variables, the squared inputs, and squared change in inputs. Choosing the parameters for the reward function (i.e., choosing the entries of the matrices Q_t , R_t used by DDP) is difficult and tedious to do by hand. Intuitively, the reward parameters tell DDP how to “trade off” between the various errors. Selecting this trade-off improperly can result in some errors becoming too large (allowing the helicopter to veer off into poorly modeled parts of the state space), or other errors being regulated too aggressively (resulting in large, unsafe control outputs).

This problem is more troublesome when using infeasible target trajectories. For instance, for the aerobatic flips and rolls performed previously in Abbeel,² a hand-coded target trajectory was used. That trajectory was not feasible, since it assumed that the helicopter could remain exactly fixed in space during the flip. Thus, there is always a (large) non-zero error during the maneuver. In this case, the particular choice of reward parameters becomes critical, since they specify how the controller should balance errors throughout the flight.

Trajectories learned from demonstration using the methods presented in Section 4, however, are generally quite close to feasible for the real helicopter. Thus, in contrast to our prior work, the choice of trade-offs is less crucial when using

these learned trajectories. Indeed, in our recent experiments it appears that a wide range of parameters work well with trajectories learned from demonstration.^h Nonetheless, when the need to make adjustments to these parameters arises, it is useful to be able to learn the necessary parameters, rather than tune them by mere trial and error.

Since we have expert demonstrations of the desired behavior (namely, following the trajectory) we can alleviate the tuning problem by employing the apprenticeship learning via inverse reinforcement learning algorithm⁴ to select appropriate parameters for our quadratic reward function. In practice, in early iterations (before convergence) this algorithm tends to generate parameters that are dangerous to use on the real helicopter. Instead, we adjust the reward weights by hand following the philosophy, but not the strict formulation of the inverse RL algorithm. In particular: we select the feature (state error) that differed most between our autonomous flights and the expert demonstrations, and then increase or decrease the corresponding quadratic penalties to bring the autonomous performance closer to that of the expert with each iteration.ⁱ Using this procedure, we obtain a good reward function in a small number of trials in practice.

We used this methodology to successfully select reward parameters to perform the flips and rolls in Abbeel,² and continue to use this methodology as a guide in selecting reward parameters.

6. EXPERIMENTAL RESULTS

6.1. Experimental setup

For our experiments we have used two different autonomous helicopters. The experiments presented here were performed with an XCell Tempest helicopter (Figure 3), but we have also conducted autonomous aerobatic flights using a Synergy N9. Both of these helicopters are capable of professional, competition-level maneuvers. We instrumented our helicopters with a Microstrain 3DM-GX1 orientation sensor. A ground-based camera system measures the helicopter’s position. A Kalman filter uses these measurements to track the helicopter’s position, velocity, orientation, and angular rate.

We collected multiple demonstrations from our expert for a variety of aerobatic trajectories: continuous in-place flips and rolls, a continuous tail-down “tic toc,” and an airshow, which consists of the following maneuvers in rapid sequence: split-S, snap roll, stall-turn, loop, loop with pirouette, stall-turn with pirouette, “hurricane” (fast backward funnel), knife-edge, flips and rolls, tic-toc, and inverted hover.

We use a large, previously collected corpus of hovering, horizontal flight, and mixed aerobatic flight data to build a crude dynamics model using the method of Section 3. This model and the pilot demonstrations are then provided to the trajectory learning algorithm of Section 4. Our trajectory

^g The 2 s horizon is a limitation imposed by available computing power. Our receding horizon DDP controller executes at 20 Hz.

^h It is often sufficient to simply choose parameters that rescale the various reward features to have approximately the same magnitude.

ⁱ For example, if our controller consistently uses larger controls than the expert but achieves lower position error, we would increase the control penalty and decrease the position penalty.

Figure 3: Our XCell Tempest autonomous helicopter.



learning algorithm includes bias terms, β^* , for each of the predicted accelerations, and hence will learn a time-dependent acceleration that is added to the crude base model. We also include terms to model position drift in the pilot demonstrations, and incorporate our prior knowledge that flips and rolls should remain roughly in place, and that maneuvers like loops should be flown in a plane (i.e., they should look flat when viewed from the top).¹²

6.2. Trajectory learning results

Figure 4(a) shows the horizontal and vertical position of the helicopter during the two loops flown during the airshow performed by our pilot. The colored lines show the expert pilot's demonstrations. The black dotted line shows the inferred ideal path produced by our algorithm. The loops are more rounded and more consistent in the inferred ideal path. We did not incorporate any prior knowledge to this effect. Figure 4(b) shows a top-down view of the same demonstrations and inferred trajectory. This view shows that the algorithm successfully inferred a trajectory that lies in a vertical plane, while obeying the system dynamics, as a result of the included prior knowledge.

Figure 4(c) shows one of the bias terms, namely the prediction errors made by our crude model for the z-axis acceleration of the helicopter for each of the demonstrations (plotted as a function of time). Figure 4(d) shows the result after alignment (in color) as well as the inferred acceleration error (black dotted). We see that the bias measurements

allude to errors approximately in the $-1G$ to $-2G$ range for the first 40 s of the airshow (a period that involves high-G maneuvering that is not predicted accurately by the “crude” model). However, only the aligned biases precisely show the magnitudes and locations of these errors along the trajectory. The alignment allows us to build our ideal trajectory based upon a much more accurate model that is tailored to match the dynamics observed in the demonstrations.

6.3. Flight results

After constructing the idealized trajectories and models using our algorithms, we attempted to fly the trajectories on the actual helicopter. As described in Section 5, we use a receding-horizon DDP controller.¹⁵ Our trajectory learning algorithm provides us with desired state and control trajectories, as well as an accurate, time-varying dynamics model tailored to the trajectory. These are provided to our DDP implementation along with quadratic reward weights chosen previously using the method described in Section 5.2. The quadratic reward function penalizes deviation from the target trajectory, s_t^* , as well as deviation from the desired controls, u_t^* , and the desired control velocities, $u_{t+1}^* - u_t^*$.

We compare the result of this procedure first with the former state of the art in aerobatic helicopter flight, namely the in-place rolls and flips of Abbeel.² That work used a single crude model, developed using the method of Section 3, along with hand-specified target trajectories, and reward weights tuned using the methodology in Section 5.2.

Figure 5(a) shows the Y-Z position^j and the collective (thrust) control inputs for the in-place rolls performed by the controller in Abbeel² and our controller using receding-horizon DDP and the outputs of our trajectory learning algorithm. Our new controller achieves (i) better position performance and (ii) lower overall collective control values (which roughly represents the amount of energy being used to fly the maneuver).

Similarly, Figure 5(b) shows the X-Z position and the collective control inputs for the in-place flips for both controllers. Like for the rolls, we see that our controller significantly outperforms the previous approach, both in position accuracy and in control energy expended.

^j These are the position coordinates projected into a plane orthogonal to the axis of rotation.

Figure 4: Colored lines: demonstrations. Black dotted line: trajectory inferred by our algorithm. (See text for details.)

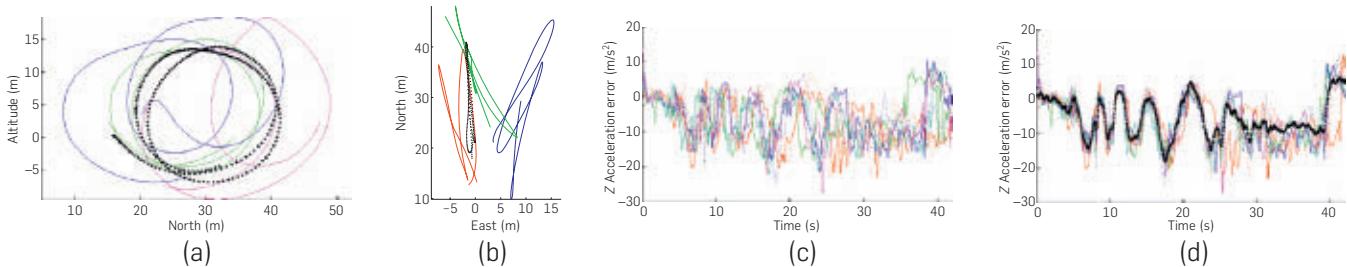
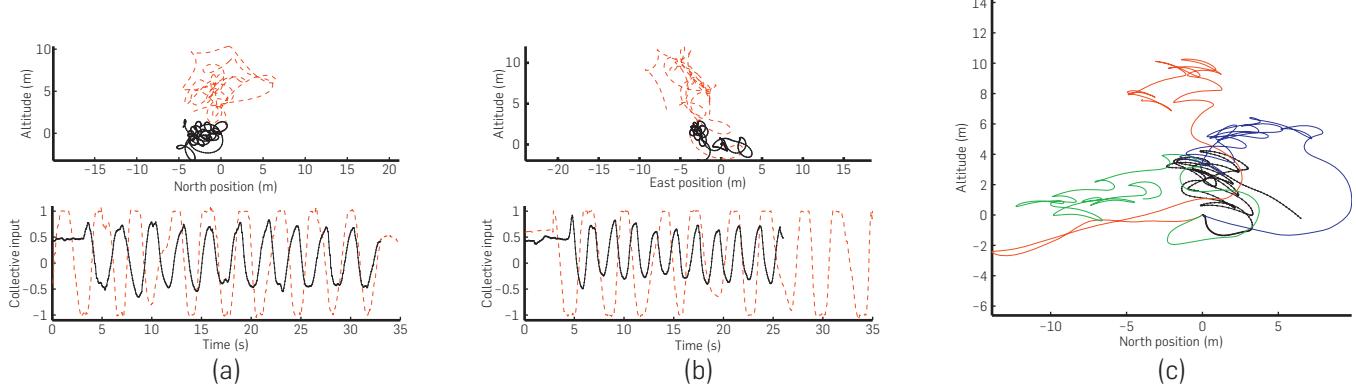


Figure 5: Flight results. (a, b) Solid black: results with trajectory learning algorithm. Dashed red: results with hand-coded trajectory from Abbeel.² (c) Dotted black: autonomous tic-toc. Solid colored: expert demonstrations.



Besides flips and rolls, we also performed autonomous “tic tocs”—widely considered to be an even more challenging aerobatic maneuver. During the (tail-down) tic-toc maneuver the helicopter pitches quickly backward and forward in place with the tail pointed toward the ground (resembling an inverted clock pendulum). The complex relationship between pitch angle, horizontal motion, vertical motion, and thrust makes it extremely difficult to create a feasible tic-toc trajectory by hand. Our attempts to use such a hand-coded trajectory, following the previous approach in Abbeel,² failed repeatedly. By contrast, the trajectory learning algorithm readily yields an excellent feasible trajectory that was successfully flown on the first attempt. Figure 5(c) shows the expert trajectories (in color), and the autonomously flown tic-toc (black dotted). Our controller significantly outperforms the expert’s demonstrations.

We also applied our algorithm to successfully fly a complete aerobatic airshow, as described in Section 6.1.

The trajectory-specific models typically capture the dynamics well enough to fly all the aforementioned maneuvers reliably. Since our computer controller flies the trajectory very consistently, however, this allows us to repeatedly acquire data from the same vicinity of the target trajectory on the real helicopter. Thus, we can incorporate this flight data into our model, allowing us to improve flight accuracy even further. For example, during the first autonomous airshow our controller achieves an RMS position error of 3.29 m, and this procedure improved performance to 1.75 m RMS position error.

Videos of all our flights are available at: <http://heli.stanford.edu>

7. CONCLUSION

We have presented learning algorithms that take advantage of expert demonstrations to successfully fly autonomous helicopters at the level of an expert human pilot. In particular, we have shown how to (i) build a rough global model from demonstration data, (ii) approximately infer the expert’s ideal desired trajectory, (iii) learn

accurate, trajectory-specific local models suitable for high-performance control, and (iv) build control systems using the outputs of our trajectory learning algorithm. Our experiments demonstrated that this design pipeline enables our controllers to fly extreme aerobatic maneuvers. Our results have shown that our system not only significantly outperforms the previous state of the art, but even outperforms our own expert pilot on a wide variety of difficult maneuvers.

Acknowledgments

We thank Garrett Oku for piloting and building our helicopters. Adam Coates is supported by a Stanford Graduate Fellowship. This work was also supported in part by the DARPA Learning Locomotion program under contract number FA8650-05-C-7261. □

References

1. Abbeel, P., Coates, A., Hunter, T., Ng, A.Y. Autonomous autorotation of an RC helicopter. *ISER 11* (2008).
2. Abbeel, P., Coates, A., Quigley, M., Ng, A.Y. An application of reinforcement learning to aerobatic helicopter flight. *NIPS 19* (2007), 1–8.
3. Abbeel, P., Ganapathi, V., Ng, A. Learning vehicular dynamics, with application to modeling helicopters. *NIPS 18* (2006), 1–8.
4. Abbeel, P., Ng, A.Y. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of ICML* (2004).
5. Abbeel, P., Quigley, M., Ng, A.Y. Using inaccurate models in reinforcement learning. In *Proceedings of ICML* (2006), ACM, NY, 1–8.
6. An, C.H., Atkeson, C.G., Hollerbach, J.M. *Model-Based Control of a Robot Manipulator*. MIT Press, 1988.
7. Anderson, B., Moore, J. *Optimal Control: Linear Quadratic Methods*. Prentice-Hall, 1989.
8. Atkeson, C., Schaal, S. Robot learning from demonstration. In *Proceedings of ICML* (1997).
9. Bagnell, J., Schneider, J. Autonomous helicopter control using reinforcement learning policy search methods. In *IEEE International Conference on Robotics and Automation* (2001).
10. Boutilier, C., Friedman, N., Goldszmidt, M., Koller, D. Context-specific independence in Bayesian networks. In *Proceedings of UAI* (1996).
11. Calinon, S., Guenter, F., Billard, A. On learning, representing and generalizing a task in a humanoid robot. In *IEEE Transactions on Systems, Man and Cybernetics, Part B*, volume 37, 2007.
12. Coates, A., Abbeel, P., Ng, A.Y. Learning for control from multiple demonstrations. In *Proceedings of ICML* (2008), 144–151.
13. Dempster, A.P., Laird, N.M., Rubin, D.B. Maximum likelihood from incomplete data via the EM algorithm. *J. Roy. Stat. Soc.* (1977).
14. Gavrillets, V., Martinos, I., Mettler, B., Feron, E. Control logic for automated aerobatic flight of miniature helicopter. In *AIAA Guidance,*

- Navigation and Control Conference* (2002).
15. Jacobson, D.H., Mayne, D.Q. *Differential Dynamic Programming*. Elsevier, 1970.
 16. La Civita, M. *Integrated Modeling and Robust Control for Full-Envelope Flight of Robotic Helicopters*. PhD thesis, Carnegie Mellon University, 2003.
 17. La Civita, M., Papageorgiou, G., Messner, W.C., Kanade, T. Design and flight testing of a high-bandwidth H_∞ loop shaping controller for a robotic helicopter. *J. Guid. Control. Dynam.*, 29, 2 (Mar.–Apr. 2006), 485–494.
 18. Leishman, J. *Principles of Helicopter Aerodynamics*. Cambridge University Press, 2000.
 19. Listgarten, J. *Analysis of Sibling Time Series Data: Alignment and Difference Detection*. PhD thesis, University of Toronto, 2006.
 20. Listgarten, J., Neal, R.M., Roweis, S.T., Emili, A. Multiple alignment of continuous time series. In *NIPS 17* (2005).
 21. Needleman, S., Wunsch, C. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Mol. Biol.*, 1970.
 22. Neu, G., Szepesvari, C. Apprenticeship learning using inverse reinforcement learning and gradient methods. In *Proceedings of UAI* (2007).
 23. Ng, A.Y., Coates, A., Diel, M., Ganapathi, V., Schulte, J., Tse, B., Berger, E., Liang, E. Autonomous inverted helicopter flight via reinforcement learning. In *ISER* (2004).
 24. Ng, A.Y., Kim, H.J., Jordan, M., Sastry, S. Autonomous helicopter flight via reinforcement learning. In *NIPS 16* (2004).
 25. Ng, A.Y., Russell, S. Algorithms for inverse reinforcement learning. In *Proceedings of ICML* (2000).
 26. Ramachandran, D., Amir, E. Bayesian inverse reinforcement learning. In *Proceedings of IJCAI* (2007).
 27. Ratliff, N., Bagnell, J., Zinkevich, M. Maximum margin planning. In *Proceedings of ICML* (2006).
 28. Roberts, J.M., Corke, P.I., Buskey, G. Low-cost flight control system for a small autonomous helicopter. In *IEEE International Conference on Robotics and Automation* (2003).
 29. Sakoe, H., Chiba, S. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing* (1978).
 30. Saripalli, S., Montgomery, J., Sukhatme, G. Visually-guided landing of an unmanned aerial vehicle, 2003.
 31. Seddon, J. *Basic Helicopter Aerodynamics*. AIAA Education Series, American Institute of Aeronautics and Astronautics, 1990.
 32. Syed, U., Schapire, R.E. A game-theoretic approach to apprenticeship learning. In *NIPS 20* (2008).

Adam Coates (acoates@cs.stanford.edu), Computer Science Department, Stanford University, Stanford, CA.

Pieter Abbeel (pabbeel@eecs.berkeley.edu), Computer Science Division, University of California, Berkeley, CA.

© 2009 ACM 0001-0782/09/0600 \$10.00

Andrew Y. Ng (ang@cs.stanford.edu), Computer Science Department, Stanford University, Stanford, CA.

Take Advantage of ACM's Lifetime Membership Plan!

- ◆ **ACM Professional Members** can enjoy the convenience of making a single payment for their entire tenure as an ACM Member, and also be protected from future price increases by taking advantage of **ACM's Lifetime Membership** option.
- ◆ **ACM Lifetime Membership** dues may be tax deductible under certain circumstances, so becoming a Lifetime Member can have additional advantages if you act before the end of 2009. (Please consult with your tax advisor.)
- ◆ Lifetime Members receive a certificate of recognition suitable for framing, and enjoy all of the benefits of **ACM Professional Membership**.

Learn more and apply at:
<http://www.acm.org/life>



Association for
Computing Machinery

Advancing Computing as a Science & Profession

Technical Perspective

A Compiler's Story

By Greg Morrisett

IN THE EARLY 1970s, pioneers like Floyd, Dijkstra, and Hoare argued that programs should be formally specified and proven correct. But for the past 40 years, most of the computer science community has discounted this vision as an unrealistic goal. Perhaps the most important reason has been simple economics: Throughout the 1980s and 1990s, the industry tended to be more interested in factors such as time-to-market than issues involving correctness.

But the context has changed dramatically over the intervening years: Security and reliability have become key concerns in the fastest growing sectors like embedded systems where lives may be at stake. Even in non-critical domains, developers must worry about bugs, including buffer overruns and race conditions that can lead to security exploits. Researchers have developed a variety of tools, including strong static type checkers, software model checkers, and abstract interpreters, all of which can (and are) used to help enforce a range of safety properties. Consequently, formal methods are in wide use today, albeit disguised within tools.

However, these tools typically operate at the source-level (or, at best, a VM bytecode level), and not at machine code level. To scale, they must make assumptions about how a compiler will refine the source code to machine code. For example, even though the C language specification does not specify an order of evaluation for function arguments, most analysis tools assume the compiler will use a left-to-right strategy, since analyzing all possible evaluation strategies would take too much time. This mismatch of assumptions, or a bug in the compiler itself, can easily render the analysis tool useless.

In the following paper, Xavier Leroy addresses these problems by describing a compiler he built and verified using the Coq proof development system. Although he is not the first

to build a verified translator, Leroy's compiler is notable and exciting for three key reasons: First, it maps a useful source language (a significant subset of C) to PowerPC assembly, making it directly usable for a range of embedded applications. Second, his compiler includes a number of analyses and optimizations, such as liveness analysis and graph-coloring register allocation, that makes the resulting code competitive with `gcc -O`. Third, the proof of correctness is mechanically checked, yielding the highest level of assurance we can hope to achieve. In short, developers can be assured that, in spite of optimization, the output of Leroy's compiler will behave the same as the source code input.

There are a number of hidden contributions to this work, beyond the construction of a fully verified, optimizing compiler: The compiler was built in a modular, pipelined fashion as a series of translations between well-specified intermediate languages, making it possible to add new optimizations or reuse components for other projects. For example, the specification for the C-subset can be reused to build new verified tools, such as a source-level analysis.

The compiler also demonstrates judicious use of *translation validation* in lieu of full verification. With translation validation, we can use unverified components to compute something (for example, an assignment of variables to registers) and need only check the output is valid (no interfering variables are assigned to the same register). Only the checker must be verified to ensure soundness, and this is often much easier than validating a full analysis and transformation. Translation validation is one engineering technique that can drastically reduce the burden of building verified systems.

This paper also shows how far we must go before verification becomes routinely feasible for production compilers or any other setting. Foremost is

the cost of constructing and maintaining the proof as the code evolves. Leroy's proof of correctness is about five to six times as big as the compiler itself, making it difficult to significantly change the code without breaking the proof. However, the proof was constructed largely by hand, and for the most part, does not take advantage of semi-automated decision procedures or proof search, a research area that has seen tremendous progress over the past decade. Indeed, work by other researchers following Leroy's lead suggests we can potentially cut the size of the proofs by up to an order of magnitude.

Perhaps the biggest challenge we face is specification. Compilers have a fairly clean notion of "correctness" (the output code should behave the same as the input code), but most software systems do not. For example, what does it mean for an operating system or Web browser to be correct? At best we can hope to formalize some safety and security properties that these systems should obey, and be willing to adapt these properties as our understanding of failures and attacks improves. In turn, this demands a verification architecture that allows specifications to be modified and adapted almost as frequently as the code. Fortunately, verified compilers make it possible to do this sort of adaptation using high-level languages without sacrificing assurance for the generated machine code.

Consequently, I think we are on the verge of a new engineering paradigm for safety- and security-critical software systems, where we rely upon formal, machine-checked verification for certification, instead of human audits. Leroy's compiler is an impressive step toward this goal. C

Greg Morrisett is the Allen B. Cutting Professor of Computer Science and associate dean for Computer Science and Engineering at Harvard University.

Formal Verification of a Realistic Compiler

By Xavier Leroy

Abstract

This paper reports on the development and formal verification (proof of semantic preservation) of CompCert, a compiler from Clight (a large subset of the C programming language) to PowerPC assembly code, using the Coq proof assistant both for programming the compiler and for proving its correctness. Such a verified compiler is useful in the context of critical software and its formal verification: the verification of the compiler guarantees that the safety properties proved on the source code hold for the executable compiled code as well.

1. INTRODUCTION

Can you trust your compiler? Compilers are generally assumed to be semantically transparent: the compiled code should behave as prescribed by the semantics of the source program. Yet, compilers—and especially optimizing compilers—are complex programs that perform complicated symbolic transformations. Despite intensive testing, bugs in compilers do occur, causing the compilers to crash at compile-time or—much worse—to silently generate an incorrect executable for a correct source program.

For low-assurance software, validated only by testing, the impact of compiler bugs is low: what is tested is the executable code produced by the compiler; rigorous testing should expose compiler-introduced errors along with errors already present in the source program. Note, however, that compiler-introduced bugs are notoriously difficult to expose and track down. The picture changes dramatically for safety-critical, high-assurance software. Here, validation by testing reaches its limits and needs to be complemented or even replaced by the use of formal methods such as model checking, static analysis, and program proof. Almost universally, these formal verification tools are applied to the source code of a program. Bugs in the compiler used to turn this formally verified source code into an executable can potentially invalidate all the guarantees so painfully obtained by the use of formal methods. In future, where formal methods are routinely applied to source programs, the compiler could appear as a weak link in the chain that goes from specifications to executables. The safety-critical software industry is aware of these issues and uses a variety of techniques to alleviate them, such as conducting manual code reviews of the generated assembly code after having turned all compiler optimizations off. These techniques do not fully address the issues, and are costly in terms of development time and program performance.

An obviously better approach is to apply formal methods to the compiler itself in order to gain assurance that it

preserves the semantics of the source programs. For the last 5 years, we have been working on the development of a *realistic, verified* compiler called CompCert. By *verified*, we mean a compiler that is accompanied by a machine-checked proof of a semantic preservation property: the generated machine code behaves as prescribed by the semantics of the source program. By *realistic*, we mean a compiler that could realistically be used in the context of production of critical software. Namely, it compiles a language commonly used for critical embedded software: neither Java nor ML nor assembly code, but a large subset of the C language. It produces code for a processor commonly used in embedded systems: we chose the PowerPC because it is popular in avionics. Finally, the compiler must generate code that is efficient enough and compact enough to fit the requirements of critical embedded systems. This implies a multipass compiler that features good register allocation and some basic optimizations.

Proving the correctness of a compiler is by no ways a new idea: the first such proof was published in 1967¹⁶ (for the compilation of arithmetic expressions down to stack machine code) and mechanically verified in 1972.¹⁷ Since then, many other proofs have been conducted, ranging from single-pass compilers for toy languages to sophisticated code optimizations.⁸ In the CompCert experiment, we carry this line of work all the way to end-to-end verification of a complete compilation chain from a structured imperative language down to assembly code through eight intermediate languages. While conducting the verification of CompCert, we found that many of the nonoptimizing translations performed, while often considered obvious in the compiler literature, are surprisingly tricky to formally prove correct.

This paper gives a high-level overview of the CompCert compiler and its mechanized verification, which uses the Coq proof assistant.^{3,7} This compiler, classically, consists of two parts: a front-end translating the Clight subset of C to a low-level, structured intermediate language called Cminor, and a lightly optimizing back-end generating PowerPC assembly code from Cminor. A detailed description of Clight can be found in Blazy and Leroy⁵; of the compiler front-end in Blazy et al.⁴; and of the compiler back-end in Leroy.^{11,13} The complete source code of the Coq development, extensively commented, is available on the Web.¹²

The remainder of this paper is organized as follows. Section 2 compares and formalizes several approaches to establishing trust in the results of compilation. Section 3

A previous version of this paper was published in *Proceedings of the 33rd Symposium on the Principles of Programming Languages*. ACM, NY, 2006.

describes the structure of the CompCert compiler, its performance, and how the Coq proof assistant was used not only to prove its correctness but also to program most of it. By lack of space, we will not detail the formal verification of every compilation pass. However, Section 4 provides a technical overview of such a verification for one crucial pass of the compiler: register allocation. Finally, Section 5 presents preliminary conclusions and directions for future work.

2. APPROACHES TO TRUSTED COMPIRATION

2.1. Notions of semantic preservation

Consider a source program S and a compiled program C produced by a compiler. Our aim is to prove that the semantics of S was preserved during compilation. To make this notion of semantic preservation precise, we assume given semantics for the source and target languages that associate observable behaviors B to S and C . We write $S \Downarrow B$ to mean that program S executes with observable behavior B . The behaviors we observe in CompCert include termination, divergence, and “going wrong” (invoking an undefined operation that could crash, such as accessing an array out of bounds). In all cases, behaviors also include a trace of the input–output operations (system calls) performed during the execution of the program. Behaviors therefore reflect accurately what the user of the program, or more generally the outside world the program interacts with, can observe.

The strongest notion of semantic preservation during compilation is that the source program S and the compiled code C have exactly the same observable behaviors:

$$\forall B, S \Downarrow B \Leftrightarrow C \Downarrow B \quad (1)$$

Notion (1) is too strong to be usable. If the source language is not deterministic, compilers are allowed to select one of the possible behaviors of the source program. (For instance, C compilers choose one particular evaluation order for expressions among the several orders allowed by the C specifications.) In this case, C will have fewer behaviors than S . Additionally, compiler optimizations can optimize away “going wrong” behaviors. For example, if S can go wrong on an integer division by zero but the compiler eliminated this computation because its result is unused, C will not go wrong. To account for these degrees of freedom in the compiler, we relax definition (1) as follows:

$$S \text{ safe} \Rightarrow (\forall B, C \Downarrow B \Rightarrow S \Downarrow B) \quad (2)$$

(Here, $S \text{ safe}$ means that none of the possible behaviors of S is a “going wrong” behavior.) In other words, if S does not go wrong, then neither does C ; moreover, all observable behaviors of C are acceptable behaviors of S .

In the CompCert experiment and the remainder of this paper, we focus on source and target languages that are deterministic (programs change their behaviors only in response to different inputs but not because of internal choices) and on execution environments that are deterministic as well (the inputs given to the programs are uniquely determined by their previous outputs). Under these conditions, there

exists exactly one behavior B such that $S \Downarrow B$, and similarly for C . In this case, it is easy to prove that property (2) is equivalent to

$$\forall B \notin \text{Wrong}, S \Downarrow B \Rightarrow C \Downarrow B \quad (3)$$

(Here, Wrong is the set of “going wrong” behaviors.) Property (3) is generally much easier to prove than property (2), since the proof can proceed by induction on the execution of S . This is the approach that we take in this work.

From a formal methods perspective, what we are really interested in is whether the compiled code satisfies the functional specifications of the application. Assume that these specifications are given as a predicate $\text{Spec}(B)$ of the observable behavior. We say that C satisfies the specifications, and write $C \models \text{Spec}$, if C cannot go wrong ($C \text{ safe}$) and all behaviors of B satisfy Spec ($\forall B, C \Downarrow B \Rightarrow \text{Spec}(B)$). The expected correctness property of the compiler is that it preserves the fact that the source code S satisfies the specification, a fact that has been established separately by formal verification of S :

$$S \models \text{Spec} \Rightarrow C \models \text{Spec} \quad (4)$$

It is easy to show that property (2) implies property (4) for all specifications Spec . Therefore, establishing property (2) once and for all spares us from establishing property (4) for every specification of interest.

A special case of property (4), of considerable historical importance, is the preservation of type and memory safety, which we can summarize as “if S does not go wrong, neither does C ”:

$$S \text{ safe} \Rightarrow C \text{ safe} \quad (5)$$

Combined with a separate check that S is well-typed in a sound type system, property (5) implies that C executes without memory violations. Type-preserving compilation¹⁸ obtains this guarantee by different means: under the assumption that S is well typed, C is proved to be well typed in a sound type system, ensuring that it cannot go wrong. Having proved properties (2) or (3) provides the same guarantee without having to equip the target and intermediate languages with sound type systems and to prove type preservation for the compiler.

2.2. Verified, validated, certifying compilers

We now discuss several approaches to establishing that a compiler preserves semantics of the compiled programs, in the sense of Section 2.1. In the following, we write $S \approx C$, where S is a source program and C is compiled code, to denote one of the semantic preservation properties (1) to (5) of Section 2.1.

Verified Compilers. We model the compiler as a total function Comp from source programs to either compiled code (written $\text{Comp}(S) = \text{OK}(C)$) or a compile-time error (written $\text{Comp}(S) = \text{Error}$). Compile-time errors correspond to cases where the compiler is unable to produce code, for instance if the source program is incorrect (syntax error, type error,

etc.), but also if it exceeds the capacities of the compiler. A compiler $Comp$ is said to be verified if it is accompanied with a formal proof of the following property:

$$\forall S, C, \text{Comp}(S) = \text{OK}(C) \Rightarrow S \approx C \quad (6)$$

In other words, a verified compiler either reports an error or produces code that satisfies the desired correctness property. Notice that a compiler that always fails ($\text{Comp}(S) = \text{Error}$ for all S) is indeed verified, although useless. Whether the compiler succeeds to compile the source programs of interest is not a correctness issue, but a quality of implementation issue, which is addressed by nonformal methods such as testing. The important feature, from a formal verification standpoint, is that the compiler never silently produces incorrect code.

Verifying a compiler in the sense of definition (6) amounts to applying program proof technology to the compiler sources, using one of the properties defined in Section 2.1 as the high-level specification of the compiler.

Translation Validation with Verified Validators. In the translation validation approach^{20, 22} the compiler does not need to be verified. Instead, the compiler is complemented by a *validator*: a boolean-valued function $\text{Validate}(S, C)$ that verifies the property $S \approx C$ a posteriori. If $\text{Comp}(S) = \text{OK}(C)$ and $\text{Validate}(S, C) = \text{true}$, the compiled code C is deemed trustworthy. Validation can be performed in several ways, ranging from symbolic interpretation and static analysis of S and C to the generation of verification conditions followed by model checking or automatic theorem proving. The property $S \approx C$ being undecidable in general, validators are necessarily incomplete and should reply `false` if they cannot establish $S \approx C$.

Translation validation generates additional confidence in the correctness of the compiled code, but by itself does not provide formal guarantees as strong as those provided by a verified compiler: the validator could itself be incorrect. To rule out this possibility, we say that a validator Validate is verified if it is accompanied with a formal proof of the following property:

$$\forall S, C, \text{Validate}(S, C) = \text{true} \Rightarrow S \approx C \quad (7)$$

The combination of a verified validator Validate with an unverified compiler $Comp$ does provide formal guarantees as strong as those provided by a verified compiler. Indeed, consider the following function:

```
Comp'(S)=
  match Comp(S) with
  | Error → Error
  | OK(C) → if Validate(S, C) then OK(C) else Error
```

This function is a verified compiler in the sense of definition (6). Verification of a translation validator is therefore an attractive alternative to the verification of a compiler, provided the validator is smaller and simpler than the compiler.

Proof-Carrying Code and Certifying Compilers. The proof-

carrying code (PCC) approach^{1, 19} does not attempt to establish semantic preservation between a source program and some compiled code. Instead, PCC focuses on the generation of independently checkable evidence that the compiled code C satisfies a behavioral specification $Spec$ such as type and memory safety. PCC makes use of a *certifying compiler*, which is a function $CComp$ that either fails or returns both a compiled code C and a proof π of the property $C \models Spec$. The proof π , also called a *certificate*, can be checked independently by the code user; there is no need to trust the code producer, nor to formally verify the compiler itself. The only part of the infrastructure that needs to be trusted is the client-side checker: the program that checks whether π entails the property $C \models Spec$.

As in the case of translation validation, it suffices to formally verify the client-side checker to obtain guarantees as strong as those obtained from compiler verification of property (4). Symmetrically, a certifying compiler can be constructed, at least theoretically, from a verified compiler, provided that the verification was conducted in a logic that follows the “propositions as types, proofs as programs” paradigm. The construction is detailed in Leroy.^{11, section 2}

2.3. Composition of compilation passes

Compilers are naturally decomposed into several passes that communicate through intermediate languages. It is fortunate that verified compilers can also be decomposed in this manner. Consider two verified compilers $Comp_1$ and $Comp_2$ from languages L_1 to L_2 and L_2 to L_3 , respectively. Assume that the semantic preservation property \approx is transitive. (This is true for properties (1) to (5) of Section 2.1.) Consider the error-propagating composition of $Comp_1$ and $Comp_2$:

$$\begin{aligned} \text{Comp}(S) = & \text{match } Comp_1(S) \text{ with} \\ & | \text{Error} \rightarrow \text{Error} \\ & | \text{OK}(I) \rightarrow Comp_2(I) \end{aligned}$$

It is trivial to show that this function is a verified compiler from L_1 to L_3 .

2.4. Summary

The conclusions of this discussion are simple and define the methodology we have followed to verify the CompCert compiler back-end. First, provided the target language of the compiler has deterministic semantics, an appropriate specification for the correctness proof of the compiler is the combination of definitions (3) and (6):

$$\forall S, C, B \notin \text{Wrong}, \text{Comp}(S) = \text{OK}(C) \wedge S \Downarrow B \Rightarrow C \Downarrow B$$

Second, a verified compiler can be structured as a composition of compilation passes, following common practice. However, all intermediate languages must be given appropriate formal semantics.

Finally, for each pass, we have a choice between proving the code that implements this pass or performing the transformation via untrusted code, then verifying its results using a verified validator. The latter approach can reduce the amount of code that needs to be verified.

3. OVERVIEW OF THE COMPCERT COMPILER

3.1. The source language

The source language of the CompCert compiler, called Clight,⁵ is a large subset of the C programming language, comparable to the subsets commonly recommended for writing critical embedded software. It supports almost all C data types, including pointers, arrays, struct and union types; all structured control (if/then, loops, break, continue, Java-style switch); and the full power of functions, including recursive functions and function pointers. The main omissions are extended-precision arithmetic (long long and long double); the goto statement; nonstructured forms of switch such as Duff's device; passing struct and union parameters and results by value; and functions with variable numbers of arguments. Other features of C are missing from Clight but are supported through code expansion (de-sugaring) during parsing: side effects within expressions (Clight expressions are side-effect free) and block-scoped variables (Clight has only global and function-local variables).

The semantics of Clight is formally defined in big-step operational style. The semantics is deterministic and makes precise a number of behaviors left unspecified or undefined in the ISO C standard, such as the sizes of data types, the results of signed arithmetic operations in case of overflow, and the evaluation order. Other undefined C behaviors are consistently turned into “going wrong” behaviors, such as dereferencing the null pointer or accessing arrays out of bounds. Memory is modeled as a collection of disjoint blocks, each block being accessed through byte offsets; pointer values are pairs of a block identifier and a byte offset. This way, pointer arithmetic is modeled accurately, even in the presence of casts between incompatible pointer types.

3.2. Compilation passes and intermediate languages

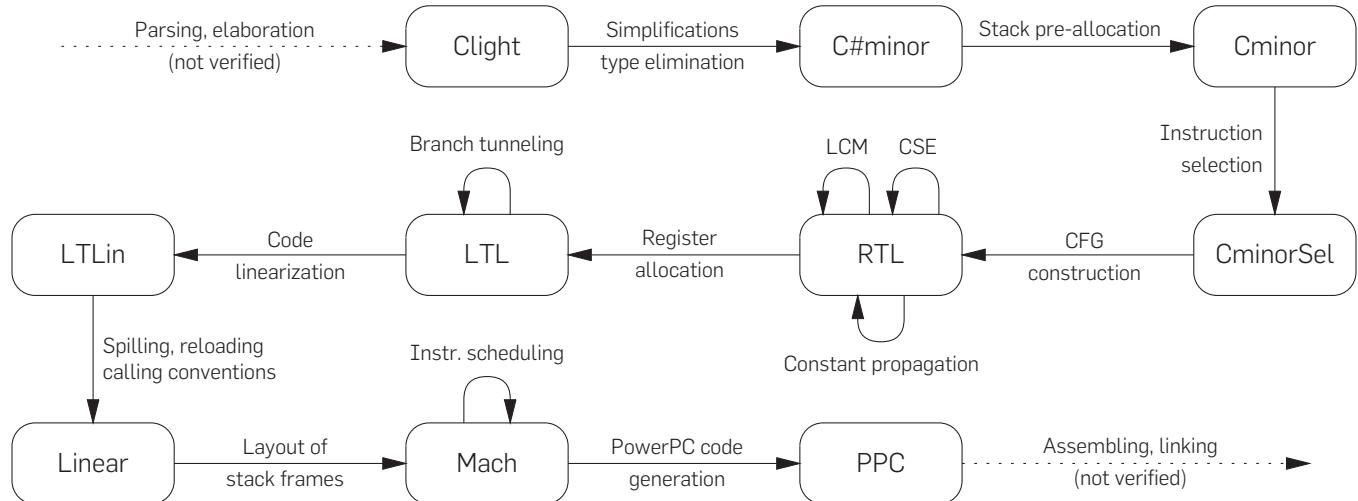
The formally verified part of the CompCert compiler translates from Clight abstract syntax to PPC abstract syntax, PPC

being a subset of PowerPC assembly language. As depicted in Figure 1, the compiler is composed of 14 passes that go through eight intermediate languages. Not detailed in Figure 1 are the parts of the compiler that are not verified yet: upstream, a parser, type-checker and simplifier that generates Clight abstract syntax from C source files and is based on the CIL library²¹; downstream, a printer for PPC abstract syntax trees in concrete assembly syntax, followed by generation of executable binary using the system’s assembler and linker.

The front-end of the compiler translates away C-specific features in two passes, going through the C#minor and Cminor intermediate languages. C#minor is a simplified, typeless variant of Clight where distinct arithmetic operators are provided for integers, pointers and floats, and C loops are replaced by infinite loops plus blocks and multilevel exits from enclosing blocks. The first pass translates C loops accordingly and eliminates all type-dependent behaviors: operator overloading is resolved; memory loads and stores, as well as address computations, are made explicit. The next intermediate language, Cminor, is similar to C#minor with the omission of the & (address-of) operator. Cminor function-local variables do not reside in memory, and their address cannot be taken. However, Cminor supports explicit stack allocation of data in the activation records of functions. The translation from C#minor to Cminor therefore recognizes scalar local variables whose addresses are never taken, assigning them to Cminor local variables and making them candidates for register allocation later; other local variables are stack-allocated in the activation record.

The compiler back-end starts with an instruction selection pass, which recognizes opportunities for using combined arithmetic instructions (add-immediate, not-and, rotate-and-mask, etc.) and addressing modes provided by the target processor. This pass proceeds by bottom-up rewriting of Cminor expressions. The target language is CminorSel, a processor-dependent variant of Cminor that offers additional operators, addressing modes, and a class

Figure 1: Compilation passes and intermediate languages.



of condition expressions (expressions evaluated for their truth value only).

The next pass translates CminorSel to RTL, a classic register transfer language where control is represented as a control-flow graph (CFG). Each node of the graph carries a machine-level instruction operating over temporaries (pseudo-registers). RTL is a convenient representation to conduct optimizations based on dataflow analyses. Two such optimizations are currently implemented: constant propagation and common subexpression elimination, the latter being performed via value numbering over extended basic blocks. A third optimization, lazy code motion, was developed separately and will be integrated soon. Unlike the other two optimizations, lazy code motion is implemented following the verified validator approach.²⁴

After these optimizations, register allocation is performed via coloring of an interference graph.⁶ The output of this pass is LTL, a language similar to RTL where temporaries are replaced by hardware registers or abstract stack locations. The CFG is then “linearized,” producing a list of instructions with explicit labels, conditional and unconditional branches. Next, spills and reloads are inserted around instructions that reference temporaries that were allocated to stack locations, and moves are inserted around function calls, prologues and epilogues to enforce calling conventions. Finally, the “stacking” pass lays out the activation records of functions, assigning offsets within this record to abstract stack locations and to saved callee-save registers, and replacing references to abstract stack locations by explicit memory loads and stores relative to the stack pointer.

This brings us to the Mach intermediate language, which is semantically close to PowerPC assembly language. Instruction scheduling by list or trace scheduling can be performed at this point, following the verified validator approach again.²³ The final compilation pass expands Mach instructions into canned sequences of PowerPC instructions, dealing with special registers such as the condition registers and with irregularities in the PowerPC instruction set. The target language, PPC, accurately models a large subset of PowerPC assembly language, omitting instructions and special registers that CompCert does not generate.

From a compilation standpoint, CompCert is unremarkable: the various passes and intermediate representations are textbook compiler technology from the early 1990s. Perhaps the only surprise is the relatively high number of intermediate languages, but many are small variations on one another: for verification purposes, it was more convenient to identify each variation as a distinct language than as different subsets of a few, more general-purpose intermediate representations.

3.3. Proving the compiler

The added value of CompCert lies not in the compilation technology implemented, but in the fact that each of the source, intermediate and target languages has formally defined semantics, and that each of the transformation and optimization passes is proved to preserve semantics in the

sense of Section 2.4.

These semantic preservation proofs are mechanized using the Coq proof assistant. Coq implements the Calculus of Inductive and Coinductive Constructions, a powerful constructive, higher-order logic which supports equally well three familiar styles of writing specifications: by functions and pattern-matching, by inductive or coinductive predicates representing inference rules, and by ordinary predicates in first-order logic. All three styles are used in the CompCert development, resulting in specifications and statements of theorems that remain quite close to what can be found in programming language research papers. In particular, compilation algorithms are naturally presented as functions, and operational semantics use mostly inductive predicates (inference rules). Coq also features more advanced logical features such as higher-order logic, dependent types and an ML-style module system, which we use occasionally in our development. For example, dependent types let us attach logical invariants to data structures, and parameterized modules enable us to reuse a generic dataflow equation solver for several static analyses.

Proving theorems in Coq is an interactive process: some decision procedures automate equational reasoning or Presburger arithmetic, for example, but most of the proofs consist in sequences of “tactics” (elementary proof steps) entered by the user to guide Coq in resolving proof obligations. Internally, Coq builds proof terms that are later rechecked by a small kernel verifier, thus generating very high confidence in the validity of proofs. While developed interactively, proof scripts can be rechecked a posteriori in batch mode.

The whole Coq formalization and proof represents 42,000 lines of Coq (excluding comments and blank lines) and approximately three person-years of work. Of these 42,000 lines, 14% define the compilation algorithms implemented in CompCert, and 10% specify the semantics of the languages involved. The remaining 76% correspond to the correctness proof itself. Each compilation pass takes between 1,500 and 3,000 lines of Coq for its specification and correctness proof. Likewise, each intermediate language is specified in 300 to 600 lines of Coq, while the source language Clight requires 1,100 lines. Additional 10,000 lines correspond to infrastructure shared between all languages and passes, such as the formalization of machine integer arithmetic and of the memory model.

3.4. Programming and running the compiler

We use Coq not only as a prover to conduct semantic preservation proofs, but also as a programming language to write all verified parts of the CompCert compiler. The specification language of Coq includes a small, pure functional language, featuring recursive functions operating by pattern-matching over inductive types (ML- or Haskell-style tree-shaped data types). With some ingenuity, this language suffices to write a compiler. The highly imperative algorithms found in compiler textbooks need to be rewritten in pure functional style. We use persistent data structures based on balanced trees, which support efficient updates without modifying data

in-place. Likewise, a monadic programming style enables us to encode exceptions and state in a legible, compositional manner.

The main advantage of this unconventional approach, compared with implementing the compiler in a conventional imperative language, is that we do not need a program logic (such as Hoare logic) to connect the compiler's code with its logical specifications. The Coq functions implementing the compiler are first-class citizens of Coq's logic and can be reasoned on directly by induction, simplifications, and equational reasoning.

To obtain an executable compiler, we rely on Coq's extraction facility,¹⁵ which automatically generates Caml code from Coq functional specifications. Combining the extracted code with hand-written Caml implementations of the unverified parts of the compiler (such as the parser), and running all this through the Caml compiler, we obtain a compiler that has a standard, cc-style command-line interface, runs on any platform supported by Caml, and generates PowerPC code that runs under MacOS X. (Other target platforms are being worked on.)

3.5. Performance

To assess the quality of the code generated by CompCert, we benchmarked it against the GCC 4.0.1 compiler at optimization levels 0, 1, and 2. Since standard benchmark suites use features of C not supported by CompCert, we had to roll our own small suite, which contains some computational kernels, cryptographic primitives, text compressors, a virtual machine interpreter and a ray tracer. The tests were run on a 2 GHz PowerPC 970 “G5” processor.

As the timings in Figure 2 show, CompCert generates code that is more than twice as fast as that generated by GCC without optimizations, and competitive with GCC at optimization levels 1 and 2. On average, CompCert code is only 7% slower than gcc -01 and 12% slower than gcc -02. The test suite is too small to draw definitive conclusions, but

these results strongly suggest that while CompCert is not going to win a prize in high performance computing, its performance is adequate for critical embedded code.

Compilation times of CompCert are within a factor of 2 of those of gcc -01, which is reasonable and shows that the overheads introduced to facilitate verification (many small passes, no imperative data structures, etc.) are acceptable.

4. REGISTER ALLOCATION

To provide a more detailed example of a verified compilation pass, we now present the register allocation pass of CompCert and outline its correctness proof.

4.1. The RTL intermediate language

Register allocation is performed over the RTL intermediate representation, which represents functions as a CFG of abstract instructions, corresponding roughly to machine instructions but operating over pseudo-registers (also called “temporaries”). Every function has an unlimited supply of pseudo-registers, and their values are preserved across function call. In the following, r ranges over pseudo-registers and l over labels of CFG nodes.

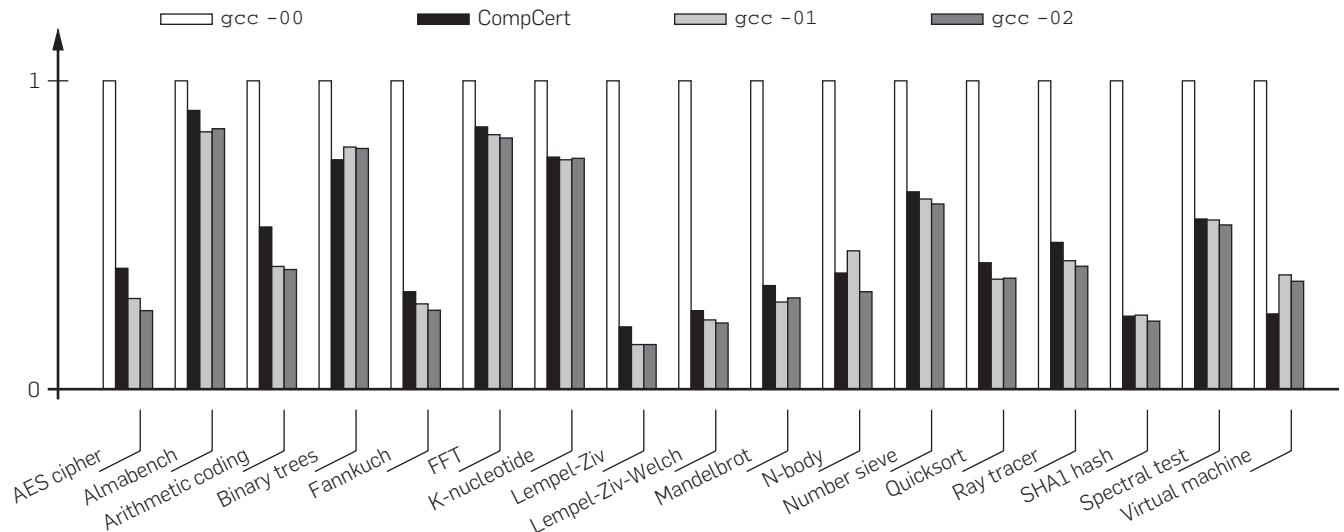
Instructions:

$i ::= \text{nop}(l)$	no operation (go to l)
$\text{op}(op, \vec{r}, r, l)$	arithmetic operation
$\text{load}(\kappa, mode, \vec{r}, r, l)$	memory load
$\text{store}(\kappa, mode, \vec{r}, r, l)$	memory store
$\text{call}(sig, (r \mid id), \vec{r}, r, l)$	function call
$\text{tailcall}(sig, (r \mid id), \vec{r})$	function tail call
$\text{cond}(cond, \vec{r}, l_{\text{true}}, l_{\text{false}})$	conditional branch
$\text{return} \mid \text{return}(r)$	function return

Control-flow graphs:

$$g ::= l \mapsto i \quad \text{finite map}$$

Figure 2: Relative execution times of compiled code.



Internal functions:

$$F ::= \{ \text{name} = id; \text{sig} = sig; \\ \text{params} = \vec{r}; \\ \text{stacksize} = n; \\ \text{entrypoint} = l; \\ \text{code} = g \}$$

parameters	size of stack data block
label of first instruction	control-flow graph

External functions:

$$Fe ::= \{ \text{name} = id; \text{sig} = sig \}$$

Each instruction takes its arguments in a list of pseudo-registers \vec{r} and stores its result, if any, in a pseudo-register r . Additionally, it carries the labels l of its possible successors. Instructions include arithmetic operations op (with an important special case $op(\text{move}, r, r', l)$ representing a register-to-register copy), memory loads and stores (of a quantity κ at the address obtained by applying addressing mode $mode$ to registers \vec{r}), conditional branches (with two successors), and function calls, tail-calls, and returns.

An RTL program is composed of a set of named functions, either internal or external. Internal functions are defined within RTL by their CFG, entry point in the CFG, and parameter registers. External functions are not defined but merely declared: they model input/output operations and similar system calls. Functions and call instructions carry signatures sig specifying the number and register classes (int or float) of their arguments and results.

The dynamic semantics of RTL is specified in small-step operational style, as a labeled transition system. The predicate $G \vdash S \xrightarrow{\cdot} S'$ denotes one step of execution from state S to state S' . The global environment G maps function pointers and names to function definitions. The trace t records the input-output events performed by this execution step: it is empty ($t = \epsilon$) for all instructions except calls to external functions, in which case t records the function name, parameters, and results of the call.

Execution states S are of the form $S(\Sigma, g, \sigma, l, R, M)$ where g is the CFG of the function currently executing, l the current program point within this function, and σ a memory block containing its activation record. The register state R maps pseudo-registers to their current values (discriminated union of 32-bit integers, 64-bit floats, and pointers). Likewise, the memory state M maps (pointer, memory quantity) pairs to values, taking overlap between multi-byte quantities into account.¹⁴ Finally, Σ models the call stack: it records pending function calls with their (g, σ, l, R) components. Two slightly different forms of execution states, call states and return states, appear when modeling function calls and returns, but will not be described here.

To give a flavor of RTL's semantics, here are two of the rules defining the one-step transition relation, for arithmetic operations and conditional branches, respectively:

$$g(l) = op(op, \vec{r}, r, l') \quad eval_op(G, \sigma, op, R(\vec{r})) = v$$

$$G \vdash S(\Sigma, g, \sigma, l, R, M) \xrightarrow{\epsilon} S(\Sigma, g, \sigma, l', R\{r \leftarrow v\}, M)$$

$$g(l) = cond(cond, \vec{r}, l_{true}, l_{false})$$

$$l' = \begin{cases} l_{true} & \text{if } eval_cond(cond, R(\vec{r})) = \text{true} \\ l_{false} & \text{if } eval_cond(cond, R(\vec{r})) = \text{false} \end{cases}$$

$$G \vdash S(\Sigma, g, \sigma, l, R, M) \xrightarrow{\epsilon} S(\Sigma, g, \sigma, l', R, M)$$

4.2. The register allocation algorithm

The goal of the register allocation pass is to replace the pseudo-registers r that appear in unbounded quantity in the original RTL code by locations l , which are either hardware registers (available in small, fixed quantity) or abstract stack slots in the activation record (available in unbounded quantity). Since accessing a hardware register is much faster than accessing a stack slot, the use of hardware registers must be maximized. Other aspects of register allocation, such as insertion of reload and spill instructions to access stack slots, are left to subsequent passes.

Register allocation starts with a standard liveness analysis performed by backward dataflow analysis. The dataflow equations for liveness are of the form

$$LV(l) = \cup \{ T(s, LV(s)) \mid s \text{ successor of } l \} \quad (8)$$

The transfer function $T(s, LV(s))$ computes the set of pseudo-registers live “before” a program point s as a function of the pseudo-registers $LV(s)$ live “after” that point. For instance, if the instruction at s is $op(op, \vec{r}, r, s')$, the result r becomes dead because it is redefined at this point, but the arguments \vec{r} become live, because they are used at this point: $T(s, LV(s)) = (LV(s)\{r\}) \cup \vec{r}$. However, if r is dead “after” ($r \notin LV(s)$), the instruction is dead code that will be eliminated later, so we can take $T(s, LV(s)) = LV(s)$ instead.

The dataflow equations are solved iteratively using Kildall's worklist algorithm. CompCert provides a generic implementation of Kildall's algorithm and of its correctness proof, which is also used for other optimization passes. The result of this algorithm is a mapping LV from program points to sets of live registers that is proved to satisfy the correctness condition $LV(l) \supseteq T(s, LV(s))$ for all s successor of l . We only prove an inequation rather than the standard dataflow equation (8) because we are interested only in the correctness of the solution, not in its optimality.

An interference graph having pseudo-registers as nodes is then built following Chaitin's rules,⁶ and proved to contain all the necessary interference edges. Typically, if two pseudo-registers r and r' are simultaneously live at a program point, the graph must contain an edge between r and r' . Interferences are of the form “these two pseudo-registers interfere” or “this pseudo-register and this hardware register interfere,” the latter being used to ensure that pseudo-registers live across a function call are not allocated to caller-save registers. Preference edges (“these two pseudo-registers should preferably be allocated the same location” or “this pseudo-register should preferably be allocated this location”) are also recorded, although they do not affect correctness of the register allocation, just its quality.

The central step of register allocation consists in coloring the interference graph, assigning to each node r a “color” $\phi(r)$ that is either a hardware register or a stack slot, under the constraint that two nodes connected by an

interference edge are assigned different colors. We use the coloring heuristic of George and Appel.⁹ Since this heuristic is difficult to prove correct directly, we implement it as unverified Caml code, then validate its results a posteriori using a simple verifier written and proved correct in Coq. Like many NP-hard problems, graph coloring is a paradigmatic example of an algorithm that is easier to validate a posteriori than to directly prove correct. The correctness conditions for the result φ of the coloring are:

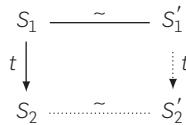
1. $\varphi(r) \neq \varphi(r')$ if r and r' interfere
2. $\varphi(r) \neq l$ if r and l interfere
3. $\varphi(r)$ and r have the same register class (int or float)

These conditions are checked by boolean-valued functions written in Coq and proved to be decision procedures for the three conditions. Compilation is aborted if the checks fail, which denotes a bug in the external graph coloring routine.

Finally, the original RTL code is rewritten. Each reference to pseudo-register r is replaced by a reference to its location $\varphi(r)$. Additionally, coalescing and dead code elimination are performed. A side-effect-free instruction $l : \text{op}(\text{op}, \vec{r}, r, l')$ or $l : \text{load}(\kappa, \text{mode}, \vec{r}, r, l')$ is replaced by a no-op $l : \text{nop}(l')$ if the result r is not live after l (dead code elimination). Likewise, a move instruction $l : \text{op}(\text{move}, r_s, r_d, l')$ is replaced by a no-op $l : \text{nop}(l')$ if $\varphi(r_d) = \varphi(r_s)$ (coalescing).

4.3. Proving semantic preservation

To prove that a program transformation preserves semantics, a standard technique used throughout the CompCert project is to show a simulation diagram: each transition in the original program must correspond to a sequence of transitions in the transformed program that have the same observable effects (same traces of input–output operations, in our case) and preserve as an invariant a given binary relation \sim between execution states of the original and transformed programs. In the case of register allocation, each original transition corresponds to exactly one transformed transition, resulting in the following “lock-step” simulation diagram:



(Solid lines represent hypotheses; dotted lines represent conclusions.) If, in addition, the invariant \sim relates initial states as well as final states, such a simulation diagram implies that any execution of the original program corresponds to an execution of the transformed program that produces exactly the same trace of observable events. Semantic preservation therefore follows.

The gist of a proof by simulation is the definition of the \sim relation. What are the conditions for two states $S(\Sigma, g, \sigma, l, R, M)$ and $S'(\Sigma', g', \sigma', l', R', M')$ to be related? Intuitively, since register allocation preserves program structure and

control flows, the control points l and l' must be identical, and the CFG g' must be the result of transforming g according to some register allocation φ as described in Section 4.2. Likewise, since register allocation preserves memory stores and allocations, the memory states and stack pointers must be identical: $M' = M$ and $\sigma' = \sigma$.

The nonobvious relation is between the register state R of the original program and the location state R' of the transformed program. Given that each pseudo-register r is mapped to the location $\varphi(r)$, we could naively require that $R(r) = R'(\varphi(r))$ for all r . However, this requirement is much too strong, as it essentially precludes any sharing of a location between two pseudo-registers whose live ranges are disjoint. To obtain the correct requirement, we need to consider what it means, semantically, for a pseudo-register to be live or dead at a program point l . A dead pseudo-register r is such that its value at point l has no influence on the program execution, because either r is never read later, or it is always redefined before being read. Therefore, in setting up the correspondence between register and location values, we can safely ignore those registers that are dead at the current point l . It suffices to require the following condition:

$$R(r) = R'(\varphi(r)) \text{ for all pseudo-registers } r \text{ live at point } l.$$

Once the relation between states is set up, proving the simulation diagram above is a routine case inspection on the various transition rules of the RTL semantics. In doing so, one comes to the pleasant realization that the dataflow inequations defining liveness, as well as Chaitin’s rules for constructing the interference graph, are the minimal sufficient conditions for the invariant between register states R, R' to be preserved in all cases.

5. CONCLUSIONS AND PERSPECTIVES

The CompCert experiment described in this paper is still ongoing, and much work remains to be done: handle a larger subset of C (e.g. including `goto`); deploy and prove correct more optimizations; target other processors beyond PowerPC; extend the semantic preservation proofs to shared-memory concurrency, etc. However, the preliminary results obtained so far provide strong evidence that the initial goal of formally verifying a realistic compiler can be achieved, within the limitations of today’s proof assistants, and using only elementary semantic and algorithmic approaches. The techniques and tools we used are very far from perfect—more proof automation, higher-level semantics and more modern intermediate representations all have the potential to significantly reduce the proof effort—but good enough to achieve the goal.

Looking back at the results obtained, we did not completely rule out all uncertainty concerning the correctness of the compiler, but reduced the problem of trusting the whole compiler down to trusting the following parts:

1. The formal semantics for the source (Clight) and target (PPC) languages.
2. The parts of the compiler that are not verified yet: the

- CIL-based parser, the assembler, and the linker.
3. The compilation chain used to produce the executable for the compiler: Coq's extraction facility and the Caml compiler and run-time system. (A bug in this compilation chain could invalidate the guarantees obtained by the correctness proof.)
 4. The Coq proof assistant itself. (A bug in Coq's implementation or an inconsistency in Coq's logic could falsify the proof.)

Issue (4) is probably the least concern: as Hales argues,¹⁰ proofs mechanically checked by a proof assistant that generates proof terms are orders of magnitude more trustworthy than even carefully hand-checked mathematical proofs.

To address concern (3), ongoing work within the CompCert project studies the feasibility of formally verifying Coq's extraction mechanism as well as a compiler from Mini-ML (the simple functional language targeted by this extraction) to Cminor. Composed with the CompCert back-end, these efforts could eventually result in a trusted execution path for programs written and verified in Coq, like CompCert itself, therefore increasing confidence further through a form of bootstrapping.

Issue (2) with the unverified components of CompCert can obviously be addressed by reimplementing and proving the corresponding passes. Semantic preservation for a parser is difficult to define, let alone prove: what is the semantics of the concrete syntax of a program, if not the semantics of the abstract syntax tree produced by parsing? However, several of the post-parsing elaboration steps performed by CIL are amenable to formal proof. Likewise, proving the correctness of an assembler and linker is feasible, if unexciting.

Perhaps the most delicate issue is (1): how can we make sure that a formal semantics agrees with language standards and common programming practice? Since the semantics in question are small relative to the whole compiler, manual reviews by experts, as well as testing conducted on executable forms of the semantics, could provide reasonable (but not formal) confidence. Another approach is to prove connections with alternate formal semantics independently developed, such as the axiomatic semantics that underline tools for deductive verification of programs (see Appel and Blazy² for an example). Additionally, this approach constitutes a first step towards a more ambitious, long-term goal: the certification, using formal methods, of the verification tools, code generators, compilers and run-time systems that participate in the development, validation and execution of critical software.

Acknowledgments

The author thanks S. Blazy, Z. Dargaye, D. Doligez, B. Grégoire, T. Moniot, L. Rideau, and B. Serpette for their contributions to the CompCert development, and A. Appel, Y. Bertot, E. Ledinot, P. Letouzey, and G. Necula for their suggestions, feedback, and help. This work was supported by Agence Nationale de la Recherche, grant number ANR-05-SSIA-0019.

References

1. Appel, A.W. Foundational proof-carrying code. In *Logic in Computer Science 2001* (2001), IEEE, 247–258.
2. Appel, A.W., Blazy, S. Separation logic for small-step Cminor. In *Theorem Proving in Higher Order Logics, TPHOLs 2007*, volume 4732 of *LNCS* (2007), Springer, 5–21.
3. Bertot, Y., Castérán, P. *Interactive Theorem Proving and Program Development—CoqArt: The Calculus of Inductive Constructions* (2004), Springer.
4. Blazy, S., Dargaye, Z., Leroy, X. Formal verification of a C compiler front-end. In *FM 2006: International Symposium on Formal Methods*, volume 4085 of *LNCS* (2006), Springer, 460–475.
5. Blazy, S., Leroy, X. Mechanized semantics for the C-like subset of the C language. *J. Autom. Reasoning* (2009). Accepted for publication, to appear.
6. Chaitin, G.J. Register allocation and spilling via graph coloring. In *1982 SIGPLAN Symposium on Compiler Construction* (1982), ACM, 98–105.
7. Coq development team. The Coq proof assistant. Available at <http://coq.inria.fr/>, 1989–2009.
8. Dave, M.A. Compiler verification: a bibliography. *SIGSOFT Softw. Eng. Notes* 28, 6 (2003), 2.
9. George, L., Appel, A.W. Iterated register coalescing. *ACM Trans. Prog. Lang. Syst.* 18, 3 (1996), 300–324.
10. Hales, T.C. Formal proof. *Notices AMS* 55, 11 (2008), 1370–1380.
11. Leroy, X. Formal certification of a compiler back-end: or: programming a compiler with a proof assistant. In *33rd Symposium on the Principles of Programming Languages* (2006), ACM, 42–54.
12. Leroy, X. The CompCert verified compiler, software and commented proof. Available at <http://compcert.inria.fr/>, Aug. 2008.
13. Leroy, X. A formally verified compiler back-end. arXiv:0902.2137 [cs]. Submitted, July 2008.
14. Leroy, X., Blazy, S. Formal verification of a C-like memory model and its uses for verifying program transformations. *J. Autom. Reasoning* 41, 1 (2008), 1–31.
15. Letouzey, P. Extraction in Coq: An overview. In *Logic and Theory of Algorithms, Computability in Europe, CiE 2008*, volume 5028 of *LNCS* (2008), Springer, 359–369.
16. McCarthy, J., Painter, J. Correctness of a compiler for arithmetical expressions. In *Mathematical Aspects of Computer Science*, volume 19 of *Proceedings of Symposia in Applied Mathematics* (1967), AMS, 33–41.
17. Milner, R., Weyhrauch, R. Proving compiler correctness in a mechanized logic. In *Proceedings of 7th Annual Machine Intelligence Workshop*, volume 7 of *Machine Intelligence* (1972), Edinburgh University Press, 51–72.
18. Morrisett, G., Walker, D., Crary, K., Glew, N. From System F to typed assembly language. *ACM Trans. Prog. Lang. Syst.* 21, 3 (1999), 528–569.
19. Necula, G.C. Proof-carrying code. In *24th Symposium on the Principles of Programming Languages* (1997), ACM, 106–119.
20. Necula, G.C. Translation validation for an optimizing compiler. In *Programming Language Design and Implementation 2000* (2000), ACM, 83–95.
21. Necula, G.C., McPeak, S., Rahul, S.P., Weimer, W. CIL: Intermediate language and tools for analysis and transformation of C programs. In *Compiler Construction*, volume 2304 of *LNCS* (2002), Springer, 213–228.
22. Pnueli, A., Siegel, M., Singerman, E. Translation validation. In *Tools and Algorithms for Construction and Analysis of Systems, TACAS '98*, volume 1384 of *LNCS* (1998), Springer, 151–166.
23. Tristan, J.-B., Leroy, X. Formal verification of translation validators: A case study on instruction scheduling optimizations. In *35th Symposium on the Principles of Programming Languages* (2008), ACM, 17–27.
24. Tristan, J.-B., Leroy, X. Verified validation of lazy code motion. In *Programming Language Design and Implementation 2009* (2009), ACM. To appear.

Xavier Leroy (xavier.leroy@inria.fr) INRIA
Paris-Rocquencourt, France

CAREERS

OxFORD Asset Management

Software Engineer - Central Oxford, England

OxFORD ASSET MANAGEMENT is seeking outstanding software engineers to develop automated trading strategies and systems to support them. Candidates should have a high quality degree in computer science or related discipline, several years C++/STL experience, and the ability to write high performance code without sacrificing correctness, stability or maintainability. Excellent compensation & benefits package offered.

No financial industry experience necessary. For more information see www.oxam.com/softwareengineer.pdf

University of Puerto Rico - Río Piedras

Computer Science Tenure-track position

The Department of Computer Science at the University of Puerto Rico, Río Piedras Campus, invites applications for a tenure-track position beginning August 2009. While priority will be given to applicants specializing in Algorithms, Theory

and Bioinformatics, applicants from all areas of Computer Science are encouraged to apply.

The Río Piedras Campus at the University of Puerto Rico is a Doctoral Research Intensive University (according to the Carnegie classification) and the Department of Computer Science is a growing department that emphasizes a strong commitment to both teaching and research. The Department is currently developing a doctoral program, therefore, Faculty are expected to create and teach undergraduate and graduate courses and to develop a visible research program capable of attracting external funding. Applicants must hold a Ph.D. in Computer Science (preferred) or a closely related field by the starting date. They must also display a commitment to excellence in teaching and a demonstrable potential for excellence in research. Applications from women and persons with diverse backgrounds and cultures are encouraged.

The main language of teaching is Spanish but English is accepted. However, research is expected to be disseminated in English. Therefore, it is required that candidates are fluent in one of the two languages at the time of appointment and that after three years, working knowledge is

obtained in the other language.

Screening will begin June 30, 2009, and will continue until the position is filled. Details about the Department are available at <http://ecom.uprrp.edu/>. Please submit a letter of interest, a current curriculum vita, a statement of teaching and research experience/interests, a copy of one recent representative research manuscripts, and the names and contact information of at least three references to:

Personnel Committee
Department of Computer Science
PO Box 23328
San Juan, PR 00931-3328

Ursinus College

Visiting Assistant Professor
of Computer Science

Ursinus College seeks to fill a FT one-year position in Computer Science beginning Fall 2009. PhD in Computer Science and teaching experience preferred, but not required. More info: <http://www.ursinus.edu/NetCommunity/Page.aspx?pid=2093>.



College of Engineering University of Miami, Coral Gables, Florida Faculty Openings at All Professorial Levels

The College of Engineering at the University of Miami (UM) invites applications and nominations for several tenure-track positions at all professorial levels. The College is seeking candidates with a strong record of scholarship with a focus on obtaining external funding, a demonstrated excellence in graduate and undergraduate teaching, interest in developing and implementing curricula that address multicultural issues, and a thoughtful commitment to university and professional service. For senior-level appointments, a proven record of extramural funding support is required. The College includes five academic departments (Biomedical Engineering; Civil, Architectural and Environmental Engineering; Electrical and Computer Engineering; Mechanical and Aerospace Engineering; and Industrial and Systems Engineering), 750 undergraduates, 250 graduate students, and 80 dedicated faculty, who have garnered national and international awards including election to the National Academy of Engineering. Our current recruitment effort is focused on the following areas of research and education: **Electrical and Computer Engineering** (multimedia, bioinformatics, sensors, imaging, computing, computer networks, signal processing, integrated electronics, power electronics, photonics), and **Industrial and Systems Engineering** (manufacturing engineering, automation and control, robotics, supply chain, health care, service systems, and risk and decisions).

At UM, collaboration is a hallmark of the faculty's activities, including joint research with colleagues in the Miller School of Medicine, the Rosenstiel School of Marine and Atmospheric Science, the School of Architecture, the College of Arts and Sciences, the School of Business Administration, the Frost School of Music, the School of Communication, the School of Education, the School of Law and the School of Nursing and Health Sciences.

A Ph.D. in engineering, science or a related discipline is required prior to the appointment. Qualified applicants should mail (a) a letter of interest, (b) a resume and (c) at least three (3) references to

Dr. Shihab Asfour, Associate Dean for Academics
Faculty Search Committee
College of Engineering
University of Miami
1251 Memorial Drive,
McArthur Engineering Bldg., Room 268
Coral Gables, FL 33146

The University of Miami, a private university, offers competitive salaries and a comprehensive benefits package including medical and dental benefits, tuition remission, vacation, paid holidays and much more. The University is an Equal Opportunity/Affirmative Action Employer.



ADVERTISING IN CAREER OPPORTUNITIES

How to Submit a Classified Line Ad: Send an e-mail to acmmEDIASales@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 ACM guidelines. **NO PROOFS** can be sent. Classified line ads are NOT commissionable.

Rates: \$325.00 for six lines of text, 40 characters per line. \$32.50 for each additional line after the first six. The MINIMUM is six lines.

Deadlines: Five weeks prior to the publication date of the issue (which is the first of every month). Latest deadlines: <http://www.acm.org/publications>

Career Opportunities Online: Classified and recruitment display ads receive a free duplicate listing on our website at: <http://campus.acm.org/careercenter>

Ads are listed for a period of 30 days.

For More Information Contact:

ACM Media Sales
at 212-626-0686 or
acmmEDIASales@acm.org



PROGRAM OFFICER, INFORMATION SCIENCE FOR C4ISR

(Computer Scientist/Electrical Engineer/
Mathematician/Physicist/Statistician)

The Office of Naval Research is seeking a qualified individual to plan, initiate, manage and coordinate sponsored basic/applied research, and advanced development programs and projects in the broad area of information science for C4ISR (Command, Control, Communications, Computers, Intelligence, Surveillance and Reconnaissance). This is a Civil Service position at the GS-13/14/15 level (\$86,927 - \$153,200) depending on individual qualifications.

The position requires knowledge and experience in the fundamental theories, concepts, and applications of research and technology development in the broad areas of information processing, integration, analysis and management for information and decision systems. Specific technical fields and applications include, but are not limited to, information science, computer science and engineering, computational sciences, decision science, network-centric information management, information infrastructure for command and control, all-source data fusion, computational decision making under uncertainty, and interoperable data structures for information/data analysis for decision systems.

This is a future vacancy to be announced. Interested parties should send resumes to bernadette.sterling.ctr@navy.mil. When the formal announcement is posted interested parties will be notified and advised how to apply.

U.S. CITIZENSHIP REQUIRED AN EQUAL OPPORTUNITY EMPLOYER

THE POVERTY LINE



One in six children lives like this.

Right here in America. In a family that's barely hanging on. Coming home to too little to eat. Losing hope. And too many people are doing nothing to help. You could change that. Join the numbers who care.

Go to www.povertyusa.org and get involved.



Catholic Campaign
for Human Development

For a three person household,
the poverty line is \$15,577.



Windows Kernel Source and Curriculum Materials for Academic Teaching and Research.

The Windows® Academic Program from Microsoft® provides the materials you need to integrate Windows kernel technology into the teaching and research of operating systems.

The program includes:

- **Windows Research Kernel (WRK):** Sources to build and experiment with a fully-functional version of the Windows kernel for x86 and x64 platforms, as well as the original design documents for Windows NT.
- **Curriculum Resource Kit (CRK):** PowerPoint® slides presenting the details of the design and implementation of the Windows kernel, following the ACM/IEEE-CS OS Body of Knowledge, and including labs, exercises, quiz questions, and links to the relevant sources.
- **ProjectOZ:** An OS project environment based on the SPACE kernel-less OS project at UC Santa Barbara, allowing students to develop OS kernel projects in user-mode.

These materials are available at no cost, but only for non-commercial use by universities.

For more information, visit www.microsoft.com/WindowsAcademic or e-mail compsci@microsoft.com.



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.



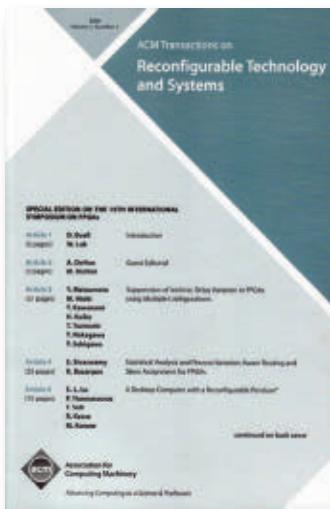
Make a difference to a student in your field.

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.

ACM Transactions on Reconfigurable Technology and Systems



This quarterly publication is a peer-reviewed and archival journal that covers reconfigurable technology, systems, and applications on reconfigurable computers. Topics include all levels of reconfigurable system abstractions and all aspects of reconfigurable technology including platforms, programming environments and application successes.



www.acm.org/trets
www.acm.org/subscribe



Association for
Computing Machinery

[CONTINUED FROM P. 120] and how it was accomplished, whereas a lot of papers in the early days were more about an implementation technique.

You've since focused your attention on distributed computing. Can you tell me about your work on fault tolerance?

As you move to a distributed environment, where you have your storage on a different machine than the one you're running on, you can end up with a system that is less reliable than before because now there are two machines, and either one of them might fail.

But there's also an opportunity for enhanced reliability. By replicating the places where you store things, you can not only guarantee they won't be lost with a much higher probability, you can also guarantee they will be available when you need them, because they're in many different places.

Tell me about Viewstamped replication, the protocol you developed for replicating data in a benign environment.

The basic idea is that, at any moment, one of the nodes is acting as what we called the primary, which means it's bossing everybody else around. If you have several different nodes, each replicating data, you need a way of coordinating them, or else you're going to wind up with an inconsistent state. The idea of the primary was that it would decide the order in which the operations should be carried out.

What happens if the primary fails?

Well, you also need a protocol—we called it the view change protocol—that allows the other replicas to elect a new leader, and you have to do that carefully to make sure everything that happened before the primary failed makes it into the next view. The nodes are constantly communicating, and they've got timers, and they can decide that a replica has failed.

Did this work lead to the protocol you subsequently developed for coping with Byzantine failures?

It did, about 10 years later. It's much harder to deal with Byzantine failures, because nodes lie, and you have to have a protocol that manages to do the right

"By replicating the places where you store things, you can not only guarantee they won't be lost with a much higher probability, you can also guarantee they will be available when you need them, because they're in many different places."

thing. My student, Miguel Castro, and I made a protocol that I can now see is sort of an extension of the original—of course, hindsight is very nice. But the primary is the boss, the other replicas are watching it, and if they feel there's a problem, they go through a view change protocol.

Recently, you've worked on the confidentiality of online storage.

If you put your data online, you want to be sure that it won't be lost. Additionally, you want to know that it isn't being leaked to third parties and that what's there is actually what you put there.

How did you get interested in the subject?

In the nineties, I did some work with my student, Andrew Meyers, on information flow control, which is a method of controlling data not by having rules about who can access it, but by having rules about what you can do with the data after you've accessed it. That's what I've been looking at recently, but the work with Andrew was programming language work, and then we just extended it. □

Leah Hoffmann is Brooklyn-based science and technology writer.

Q&A

Liskov on Liskov

Barbara Liskov talks about her groundbreaking work in data abstraction and distributed computing.

BARBARA LISKOV, a professor at the Massachusetts Institute of Technology (MIT) and winner of the 2008 ACM A.M. Turing Award, has worked throughout her career to make software systems more accessible, reliable, and secure. We caught up with her recently to discuss a few of her most important accomplishments—and to find out what she's working on now.

Let's talk about CLU, the programming language you developed in the 1970s to handle abstract data types.

Before I came to MIT, I was working on the VENUS system, and I got some ideas about a different way of modularizing programs around what I called multi-operation modules. When I came to MIT, I started to think of that in terms of data types. And then I decided the best way to continue the research was to develop a programming language.

How did your ideas differ from the research that was going on at the time?

When I started, the main way people thought about modularization was in terms of subroutines—of abstracting from how you wrote a procedure to calling that procedure, say, a sort routine, or a lookup routine. But they didn't have any way of linking a bunch of procedures together.

And that's what CLU's clusters accomplish.

Yes. A cluster would have all the operations you needed to interact with a data object, and inside you could



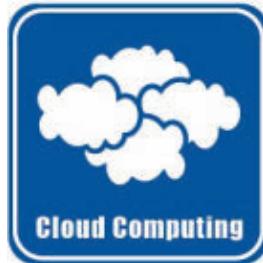
implement it and later re-implement it however you wanted.

Eventually, object-oriented programming evolved from your work on CLU.

Object-oriented programming evolved from two different strands. There was my work on data abstraction and some

related work that was going on at [Carnegie Mellon University]. The other influence was Smalltalk. Both of these were sort of getting at the same idea in slightly different ways, but the big difference between my work and the Smalltalk work was that I focused on making a very strong distinction between what a module did [CONTINUED ON P. 119]

The Technical Committee on Services Computing (TC-SVC) of IEEE Computer Society Sponsors



CALL FOR PARTICIPATION

KEYNOTES
TUTORIALS
PANELS
RESEARCH
PAPERS
INDUSTRY
PAPERS
SUMMER
SCHOOL ON
SERVICES
COMPUTING
INNOVATION
SHOWCASE

SPONSORS



LINKS

CLOUD-I 2009:
thecloudcomputing.org/2009/1
CLOUD-II 2009:
thecloudcomputing.org/2009/2
ICWS 2009: icws.org
SCC 2009:
conferences.computer.org/scc
SERVICES 2009:
servicescongress.org

2009 IEEE International Conference on Cloud Computing (CLOUD 2009)

<http://tab.computer.org/tcsc> and <http://thecloudcomputing.org>

CLOUD 2009 PART 1

Co-located with IEEE ICWS 2009
July 6-10, 2009, Los Angeles, CA, USA



CLOUD 2009 is the identified hot-topic conference by the 2009 World Congress on Services (SERVICES 2009). Part 1 of CLOUD 2009 will be co-located with the **2009 IEEE International Conference on Web Services** (ICWS 2009, <http://www.icws.org>) on July 6-10, 2009, Los Angeles, California, USA. Part 2 of CLOUD will be co-located with the **2009 IEEE International Conference on Services Computing** (SCC 2009, <http://conferences.computer.org/scc/2009>) on September 21-25, 2009, Bangalore, India.

Cloud Computing has become a scalable services delivery platform in the field of **Services Computing**. The technical foundations of Cloud Computing include Service-Oriented Architecture (SOA) and Virtualizations of hardware and software. The goal of Cloud Computing is to share resources among the cloud service consumers, cloud partners, and cloud vendors in the cloud value chain. The resource sharing at various levels results in various cloud offerings such as infrastructure cloud (e.g. hardware, IT infrastructure management), software cloud (e.g. SaaS focusing on middleware as a service, or traditional CRM as a service), application cloud (e.g. Application as a Service, UML modeling tools as a service, social network as a service), and business cloud (e.g. business process as a service).

Based on the technology foundations and industry driving forces, the 2009 International Conference on Cloud Computing (CLOUD 2009) is created to provide a prime international forum for both researchers and industry practitioners to exchange the latest fundamental advances in the state of the art and practice of Cloud Computing, identify emerging research topics, and define the future of Cloud Computing.

CLOUD 2009 PART 2

Co-located with IEEE SCC 2009
September 21-25, 2009, Bangalore, India



Contact the program chair Dr. Liang-Jie Zhang (LJ) at zhanglj@ieee.org



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.

The screenshot shows the acmqueue homepage with a navigation bar at the top. The main content area includes:

- Welcome to acmqueue**: A brief introduction to the site's mission.
- LATEST QUEUE CONTENT**: A section featuring recent articles:
 - Security in the Browser** by Brian Poole, March 19, 2009
 - Caterpillars 2.0: When the Cloud Turns Dark** by Brian Poole, March 19, 2009
 - How Do I Model That? Let Me Count the Ways** by Jeff Mier, March 17, 2009
 - Randy Vernon: Don't Be Typecast as a Software Developer** by Randy Vernon, March 13, 2009
 - A Conversation with Ven Jenkins** by Ven Jenkins, January 01, 2009
- QUEUE AUTHOR BLOG ROLL**: A list of blog posts from various authors:
 - James Gosling: Wandering the Web
 - Jeff Mier: Links for Tuesday, March 24, 2009
 - Bruce Schneier: Election Fraud in Kentucky
 - Greg Lakey: Monday, 23 March 2009
 - Gary Orlitzky: Barometric Climate for Today, Forecast: Lesser Tuna, Part 12
 - Vin Gray: More Music, Mystery
 - Darren Aspin: Threading Java's SOGo Threaded 1.0
 - James Hollingshead: AOC (Hadoop) in the Data Center

Visit today!

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